Homework 3 Solution

Problem 1:

$f: N \to N$ is a function computable in $\mathrm{TIME}(n^4)$

$g: N \to N$ " " " " " $\mathrm{TIME}(n^3)$

Show that $f(g(x))$ can be computed in $\mathrm{TIME}(n^{12})$

$\because$ g is a computable function in $\mathrm{TIME}(n^3)$

$\therefore$ For any input $x \in N$ with size n, the function $g(x)$ takes at most $O(n^3)$ steps to halt.

$\because$ $\mathrm{TIME} \subset \mathrm{SPACE}$

$\therefore$ The output of $g(x)$ will take a space of at most $O(n^3)$ squares.

$\because$ f is a computable function in $\mathrm{TIME}(n^4)$

$\therefore$ For any input $\cancel{y} \in N$, with size n, the function $f(\cancel{y})$ takes at most $O(n^4)$ steps to halt.

$\therefore$ $f(g(x))$ is f running on input $g(x)$ which has size $O(n^3)$

$\therefore$ $F(g(x))$ can be computed in $\mathrm{TIME}((n^3)^4)$

$= \mathrm{TIME}(n^{12})$

Problem 2

(i) Prove that $TIME(n^2)$ is closed under difference.

Assume A and B are 2 languages in $TIME(n^2)$
This means that we have T_A and T_B that solve A and B respectively in $TIME(n^2)$ (maximum of $O(n^2)$ steps)

We can build a TM T_{A-B} which does the following on an input $x \in \Sigma^*$:
1- Simulate T_A on x
2- If $T_A(x)$ halts and rejects then halt and reject.
3- If $T_A(x)$ accepts then simulate T_B on x.
4- If $T_B(x)$ halts and accepts, then accept
5- If $T_B(x)$ " " rejects, then reject.

From the above TM description, all steps run in $O(n^2)$ time.
Specifically $O(n^2+1+n^2+1)$

$\therefore$ A-B is also in $TIME(n^2)$
$\therefore$ $TIME(n^2)$ is closed under difference.

(ii) It's not known if NTIME(n^2) is also closed under complement.

A language L is in NTIME(n^2) if there is a NTM T that accepts it in time $O(n^2)$.
T accepts a string $x \in \Sigma^*$ if there is at least one accepting path in the set of paths it could go through.

A simple reverse of all accepting and rejecting <u>states</u> will not create a TM T' which accepts L correctly.

If $x \in L$, then $x \notin \bar{L}$
But by reversing all the rejecting paths in T to accepting paths in T', there is a chance that $T'(x)$ will also accept x although it shouldn't.

$\therefore$ It's not known if NTIME(n^2) is closed under complement.

Problem 3

(4) $A \leq_m^P B$ and $B \in P \Rightarrow A \in P$

$\because A \leq_m^P B$, then there exists a pol. time computable function f which can convert $x \in A$ to $f(x) \in B$

$\because B \in P$, then there exists a TM T_B which decides B in polynomial time.

We can build a TM T_A to decide A which on input $x \in \Sigma^*$ does the following:

1- Run $f(x)$ [compute $f(x)$]

2- Run M_B on the output of $f(x)$

3- If $M_B(f(x))$ accepts $\rightarrow$ halt and accept
   If $M_B(f(x))$ rejects $\rightarrow$ halt and reject

Steps 1 and 2 run in pol. time and step 3 takes only one step.

$\because$ The addition of polynomials is a pol.

$\therefore T_A$ runs in pol. time

$\therefore A \in P$

*NOTE: This uses the fact from problem #1 that if you compose two polynomial time functions together, the function you obtain is also in PF.

(5) $A \leq_m^p B$ and $B \in NP \Rightarrow A \in NP$

Use the same idea as the previous part. But here we use NTM which run in pol. time.

$\therefore$ We can create an NTM T_A which runs non-deterministically in pol. time.

$\therefore$ T_A is accepted in pol. time.

$\therefore$ $A \in NP$

Problem 4

P=NP iff the function MAXCLIQUE is pol. time computable.

($\rightarrow$) Assume P = NP

∵ We know CLIQUE is NP-complete

∴ CLIQUE $\in$ NP

∴ CLIQUE $\in$ P which means that we have a TM M_1 which can decide ~~whether~~ for an input (G, k) whether graph G has a clique of size k in pol. time.

We can create a TM M_2 which will decide the output of MAXCLIQUE in pol. time.

M_2 will do the following on input G:

1- Run M_1 on (G, |V|) where |V| is the total number of vertices in G.
(This could be given on the input tape, or calculated in pol. time)

2- If M_1(G, |V|) accepts (which means that there is a clique of size |V|) then halt and output |V|

3- If it rejects, then repeat step one with M_1(G, |V|-1) and check if it accepts or not.

- If it rejects, then repeat with M_1(G, |V|-2), M_1(G, |V|-3) and so on till either we reach an accept or we reach M(G, 1).

All the steps of M_2 run in pol. time and the recursion occurs for only |V| times.

$\therefore$ M_2 runs in pol. time
$\therefore$ MAXCLIQUE is pol. time computable.

($\Leftarrow$) Assume MAXCLIQUE is pol. time computable.

Then $\exists$ TM M_1 which runs MAXCLIQUE in pol. time on any input G.
We can build a TM M_2 to decide CLIQUE in pol. time. which will do the following on any input (G, k),
1- Compute MAXCLIQUE(G) to get x (or Run $M_1(G)$),
2- If $x \geqslant k$, then halt and accept
Else, halt and reject.

$\because$ M_2 runs in pol. time
$\therefore$ We can decide CLIQUE in pol. time.
$\therefore$ CLIQUE $\in P$

$\because$ We know CLIQUE is NP-complete
$\therefore$ $\forall L \in NP$, $L \leq_m^P$ CLIQUE

If CLIQUE $\in P$
$\therefore$ All $L \in NP$ will also $\in P$

$\therefore$ $NP = P$

Problem 5

Show that some infinite subset of CLIQUE belongs to P

Note: There are many solutions for this problem. This is an example.

The set of all 2-clique graphs is an infinite subset of the CLIQUE problem set

We can build a TM M which decides whether a graph G is a 2-clique graph by simply checking whether an edge exists in the graph.

M runs in polynomial-time.

∴ We have shown that the subset (G, 2) of CLIQUE belongs to P

Problem 6

Show that $L = \{F \mid F$ is a prop. formula that has atleast 2 satisfying assignments$\}$ is NP-complete.

First, we need to prove that $L \in NP$
We can build an NTM that takes an input F and chooses 2 different assignments for F non-deterministically and checks whether they satisfy F or not.
If yes, then halt and accept.
If no, then halt and reject.
∴ ~~$L \in NP$~~
∵ This NTM is ND and runs in pol. time
∴ $L \in NP$

Second, we need to show $\forall A \in NP$, $A \leq_m^p L$
To do this, we need to show that $SAT \leq_m^p L$

Define a function g which takes as an input formula F and outputs $\underbrace{F \wedge (x \vee \bar{x})}_{F'}$ where x is a new variable not in F.
g is pol. computable since it only takes pol. number of steps to scan F and add the extra part to it. *Note that $(x \vee \neg x)$ has exactly two satisfying assignments,*
Using g we can show that if $F \in SAT$, then $F' \in L$
and if $F \notin SAT$, then $F' \notin L$

∴ $SAT \leq_m^p L$
∵ SAT is NP-complete

$\therefore \forall A \in NP,\ A \leq_m^p SAT$

$\because SAT \leq_m^p L$

$\therefore \forall A \in NP,\ A \leq_m^p L$

$\therefore L$ is NP-complete.

Problem 7

Define partial function f by $f(\emptyset)$ = some satisfying assignment $\emptyset$, if exists, where $\emptyset$ is the formula of prop. logic.

Show that f is pol. time-computable iff $P = NP$

($\rightarrow$) Assume f is pol. time computable

We can build a TM M which will do the following on an input $\emptyset \in \Sigma^*$:

1- Compute $f(\emptyset)$

2- If output is null ($\perp$), then halt and reject
  Else, halt and accept.

$\because$ M runs in pol. time and decides SAT

$\therefore$ SAT $\in P$

$\because$ SAT is NP-complete

$\therefore$ $\forall L \in NP$, $L \leq_m^P$ SAT

$\therefore$ If SAT $\in P$, then $P = NP$

($\leftarrow$) Assume $P = NP$

$\therefore$ SAT $\in P$ which means that there is a pol. time TM M_1 which decides SAT.

We can use M_1 to build M_2 which computes the function f.

On input $\emptyset(x_1, x_2, \ldots, x_n)$, M_2 will do the following:

1- Run $M_1(\emptyset)$

2- If $M_1(\emptyset)$ accepts, go to step 3
  If $M_1(\emptyset)$ rejects, halt and reject with no output.

3- Run $M_1(\emptyset(0, x_2, \ldots, x_n))$

4- If accepts, run $M_1(\emptyset(0, 0, x_3, \ldots, x_n))$
  If rejects, run $M_1(\emptyset(1, 0, x_3, \ldots, x_n))$

Keep on doing this until we ~~[illegible]~~ assign correct values to the n variables.
Then output these values as the output of $M_2(\phi)$.

$M_2(\phi)$ computes f since it outputs some satisfying assignment of ϕ. and it runs in pol. time.

$\therefore$ F is pol. time computable

Problem 8

The proof is wrong because the process of converting a boolean formula F to DNF formula F' isn-t polynomial.
The conversion causes an exponential explosion of the formula thus causing the algorithm to be non-polynomial. Thus, wrong.