Complexity - Exercise 2

Oren Roth

December 18, 2017

Question 1

Let M be the TM which decide A and $p(n)$ a polynomial, which for any input with size n, M runs at most $p(n)$ steps with oracle access to inputs with length at most $n-1$. We will describe M' that will use only $n * p(n)$ space for any input of size n and decide A. This will show $A \in PSPACE$.
Given $x \in \Sigma^*$, M' will save n slots with size $p(n)$ each. M' runs as follows:

- Copy x to the first slot
- run M on the input in slot 1. When oracle request is asked copy the input for the oracle to the next slot and runs M on it.
- Continue the same process when oracle request is asked copy the input for the oracle to the next slot and runs M on it and so on.
- When M answer on input of slot $i > 1$, clear slot i and return the answer to M on slot $i-1$ that asked the question to the oracle and continue the run of M on slot $i-1$.
- When answer of M appears on slot 1, M' return the same.

First notice that any calls for oracle is shrink the input by one, so we will use at most n slots, because each slot is with size $p(n)$, M' uses $poly(n)$ space. Moreover as M need $p(n)$ time for an input in size n, it for sure use at most $p(n)$ space for inputs in size $\leq n$, in each slot we run on inputs with size $\leq$ to n and hence each slot as sufficient space for running M.
If all the calls for the oracle are correct M' is just simulating M on slot 1 and hence $L(M') = L(M)$.
Assume that all the calls for the oracle in input y with $|y| < k$ are simulating correctly by M' and let y be input for the oracle with $|y| = k$, for this we will open new slot and run M on y, with every oracle call on inputs in size at most $k-1$ which by the induction assumption we know M' simulate correctly and hence, M' on y will answer exactly as M on y.

Question 2

We will assume $CO - NP \subseteq NP$, and derive $PH \subseteq NP$ and hence $PH = NP$.

$$PH = \bigcup_{i=0}^{\infty} \Sigma_i^p$$

We will show by induction on i that $\Sigma_i^p \subseteq NP$.
$i = 0$, $\Sigma_0^p = P \subseteq NP$.

$i = 1$, $\Sigma_1^p = NP \subseteq NP$.
Assume that for i, $2 \leq i < k$:

$$\Sigma_i^p \subseteq NP$$

We know $NP \subseteq \Sigma_i^p$ and hence $NP = \Sigma_i^p$.
This makes $\Pi_i^p = CO - NP$, and because we know $CO - NP \subseteq NP$ we get $\Pi_i^p \subseteq \Sigma_i^p$.
Let us show $\Sigma_k^p \subseteq NP$:

$$\begin{aligned} &L \in \Sigma_k^p \iff \exists M \; polynomial \; TM \; s.t: \\ &x \in L \iff \exists y_1 \forall y_2 \ldots Q_k y_k \; M(x, y_1, y_2, \ldots, y_k) = 1 \end{aligned}$$

Let us define L':

$$L' = \{(x, y_1) : \forall y_2 \exists y_3 \ldots Q_k y_k \; M(x, y_1, y_2, \ldots, y_k) = 1\}$$

So we have:

$$x \in L \iff \exists y_1 \forall y_2 \ldots Q_k y_k \; M(x, y_1, y_2, \ldots, y_k) = 1 \iff \tag{1}$$
$$\forall y_2 \exists y_3 \ldots Q_k y_k \; M(x, y_1, y_2, \ldots, y_k) = 1 \iff (x, y_1) \in L' \tag{2}$$

By definition $L' \in \Pi_{k-1}^p$ but we showed for all indexes $i < k$ that $\Pi_i^p \subseteq \Sigma_i^p$, and so we conclude $\Pi_{k-1}^p \subseteq \Sigma_{k-1}^p$, and:

$$L' \in \Sigma_{k-1}^p$$

and hence there exists polynomial M' s.t.

$$(x, y_1) \in L' \iff \exists y_2 \forall y_3 \ldots Q_k y_k \; M'(x, y_1, y_2, \ldots, y_k) = 1$$

And moreover:

$$\begin{aligned} (x, y_1) \in L' \iff \exists y_2 \forall y_3 \ldots Q_k y_k \; M'(x, y_1, y_2, \ldots, y_k) = 1 \iff \\ \exists y_1, y_2 \forall y_3 \ldots Q_k y_k \; M'(x, y_1, y_2, \ldots, y_k) = 1 \end{aligned}$$

Adding with the inequality in (1) we conclude:

$$\begin{aligned} x \in L \iff \exists y_1 \; s.t. \; (x, y_1) \in L' \iff \\ \exists y_1, y_2 \forall y_3 \ldots Q_k y_k \; M'(x, y_1, y_2, \ldots, y_k) = 1 \end{aligned}$$

And we got that $L \in \Sigma_{k-1}^p$, which by IA we know $\Sigma_{k-1}^p \subseteq NP$, and we derive our desire conclusion:

$$L \in NP$$

And thus,

$$\Sigma_k^p \subseteq NP$$

By induction we showed that for any $i \in \mathbb{N}$, $\Sigma_i^p \subseteq NP$ and thus $PH \subseteq NP$, we know $NP \subseteq PH$, so we conclude: $PH = NP$.

Question 3

a.

Assume $DP \subseteq NP$, for any $L \in CO - NP$, if we will look at:

$$L \cap \Sigma^* = L$$

Because $\Sigma^* \in P \subseteq NP$ we got $L \in DP$. So we showed that also:

$$CO - NP \subseteq DP$$

Together with our assumption we got:

$$CO - NP \subseteq DP \subseteq NP$$

And we already showed in Q.2 that $CO - NP \subseteq NP$ is concluding to $PH = NP$.

b.

First I will show EC (Exact -Clique) is in DP, as we know $CLIQUE \in NP$

$$CLIQUE = \{< G, k >: \text{there exists clique in } G \text{ of size } k\}$$

So for:

$$CLIQUE_{+1} = \{< G, k >: \text{there exists clique in } G \text{ of size } k+1\}$$

Clearly $CLIQUE_{+1} \in NP$, so $\overline{CLIQUE_{+1}} \in CO - NP$ and we got:

$$EC = CLIQUE \cap \overline{CLIQUE_{+1}}$$

And we conclude $EC \in DP$.
To show $EC \in DP - hard$, given $L \in DP$ I will show how to reduce it in polynomial time to EC. I will use the known fact $CLIQUE \in NP - Complete$. From this fact we know that $\overline{CLIQUE} \in CO - NP - Complete$. Given $L \in DP$, we know:

$$L = L_1 \cap L_2 \quad where:$$
$$L_1 \in NP \quad L_2 \in CO - NP$$

And because $CLIQUE \in NP - Complete$ and $\overline{CLIQUE} \in CO - NP - Complete$, there those reductions:

$$L_1 \leq_p CLIQUE$$
$$L_2 \leq_p \overline{CLIQUE}$$

So there exists two polynomial reduction function f_1, f_2 s.t.:

$$x \in L_1 \iff f_1(x) =< G_1, k_1 >\in CLIQUE$$
$$x \in L_2 \iff f_2(x) =< G_2, k_2 >\in \overline{CLIQUE}$$

Notice: WLOG we can assume that $k_1 \neq k_2$ as if it's not the case there f_1' which will just add another node to G_1, the output of $f_1(x)$ and connect it to all the nodes, and set $k_1' = k_1 + 1$. f_1' will

be also a reduction function with the same properties and will have $k_1' \neq k_2$.

Recall that the standard reduction construction from $3-SAT$ to $CLIQUE$ are having that if there is a clique in size k_i this is the biggest clique. (As k_i is the biggest size of clique possible, if by contradiction there exists a clique of size bigger than k_i it will lead by the pigeonhole principle to two nodes from the same clause connected together). So we can adjust the previous equalities:

$$x \in L_1 \iff f_1(x) =< G_1, k_1 > \text{and the biggest clique in } G_1 \text{ is in size } k_1.$$
$$x \in L_2 \iff f_2(x) =< G_2, k_2 > \text{and the biggest clique in } G_2 \text{ is in size smaller than } k_2$$

We will set new reduction function f_i' ($i \in \{1,2\}$) on input x, f_i' will run $f_i(x)$ and receive $< G_i, k_i >$ it will add K_{k_i-1} to G_i denote it by G_i' and return $< G_i', k_i >$, where K_j is a clique in size j. Now we will have those properties:

$$x \in L_1 \Rightarrow f_1'(x) =< G_1', k_1 > \text{and the biggest clique in } G_1' \text{ is in size } k_1$$
$$x \notin L_1 \Rightarrow f_1'(x) =< G_1', k_1 > \text{and the biggest clique in } G_1' \text{ is in size } k_1 - 1$$
$$x \in L_2 \Rightarrow f_2'(x) =< G_2', k_2 > \text{and the biggest clique in } G_2' \text{ is in size } k_2 - 1$$
$$x \notin L_2 \Rightarrow f_2'(x) =< G_2', k_2 > \text{and the biggest clique in } G_2' \text{ is in size } k_2$$

Now we ready to set our reduction function f from L to EC. Given $x\Sigma^*$ we will run $f_1'(x), f_2'(x)$ and get $< G_1', k_1 >$ and $< G_2', k_2 >$ where by our notice $k_1 \neq k_2$, we return $f(x) =< G_1' \times G_2', k_1 \cdot (k_2 - 1) >$ ($G_1' \times G_2'$ will just return both union of the graphs with addition than each node in G_1' will be connected to all the nodes of G_2'). And we got:

$$x \in L \Rightarrow x \in L_1 \wedge x \in L_2 \Rightarrow$$
$$\text{the biggest clique in } G_1' \text{ is in size } k_1 \wedge$$
$$\text{the biggest clique in } G_2' \text{ is in size } k_2 - 1 \Rightarrow$$
$$\text{the biggest clique in } G_1' \times G_2' \text{ is of size } k_1 \cdot (k_2 - 1) \Rightarrow$$
$$f(x) \in EC$$

$$x \notin L \Rightarrow x \notin L_1 \vee x \notin L_2 \Rightarrow$$
$$\text{the biggest clique in } G_1' \text{ is in size } k_1 - 1 \vee$$
$$\text{the biggest clique in } G_2' \text{ is in size } k_2 \Rightarrow$$
$$\text{the biggest clique in } G_1' \times G_2' \text{ is size } ((k_1 - 1) \cdot k_2) \text{ or } (k_1 \cdot k_2) \text{ or } ((k_1 - 1) \cdot (k_2 - 1))$$
$$\text{which in any case } \neq (k_1 \cdot (k_2 - 1)) \Rightarrow$$
$$f(x) \notin EC$$

Question 4

a.

We will show algorithm that uses only logarithmic space and will decide $FVAL$. Given (F, a), for any literal in F we refer in our algorithm $a(n) \in \{T, F\}$ as the substition of a on that literal, notice that this can be checked with $O(1)$ space. In our algorithm we will think of F as a binary tree, and use the fact that we can decide for any node n in tree F which direct subtree of n is bigger in logarithmic space. This can be calling the following sub routine twice once where n is the left subtree and after setting n to be the right subtree.

```
procedure CheckSize
    counter ← 0
    level ← 0
    cameFrom ← ∅
loop:
    if n is litreal or cameFrom =′ right′ then
        counter ← counter + 1
        if level = 0 then
            return counter
        else
            level ← level − 1
            if n is a right node then
                cameFrom ←′ right′
            else
                cameFrom ←′ left′
            n ← n.parent
            goto loop
    else if cameFrom =′ left′ then
        level ← level + 1
        n ← n.rightnode
        cameFrom ← ∅
        goto loop
    else if cameFrom = ∅ then
        level ← level + 1
        n ← n.leftnode
        goto loop
```

Than we will compare between those counters and decide which subtree is bigger. We used here 4 variables which at most need $log(N)$ space where N is number of nodes in tree which is $O(n)$. We used here and in our algorithm the notation n for variable for node, we can think of n denote the subtree we are looking now as two pointers that mark the position in F, remark that each pointer doesn't need more than $log(n)$ space. We can update $n \leftarrow n.parent$ by just update the pointers to the outside parentheses.

Now we are ready to present the algorithm that will decide $FVAL$, this algorithm will basically will valuate the tree F from down to up traversing through the bigger subtree in each step:

Algorithm 1 FVAL(F,a)

```
ans, cameFrom, S ← ∅
i, level ← 0
n ← root of F
loop:
if n is litreal then
    ans ← a[n]
levelup:
    if level = 0 then return ans
    else if n is a right node then
        cameFrom ←' right'
    else
        cameFrom ←' left'
    n ← n.parent
    goto loop
else if ans = ∅ then
    if (CheckSize(n.rightnode) > CheckSize(n.leftnode)) then n ← n.rightnode
    else   n ← n.leftnode
    level ← level + 1
    if S[i] ≠ null then i ← i + 1
    goto loop
else if S[i] = null then
    S[i] ← ans
    ans ← ∅
    if cameFrom =' right' then
        n ← n.leftnode
    else
        n ← n.rightnode
    cameFrom ← ∅
    goto loop
else
    if n is from the F_1 ∧ F_2 format then
        ans ← min{ans, S[i]}
    else
        ans ← max{ans, S[i]}
    S[i] ← null
    level ← level − 1
    if i > level then i ← level
    goto levelup
```

$ans, cameFrom$ takes only $O(1)$ space, $i, level$ are representing a number which is big at most as n, so we need only $O(log(n))$ space to save them. For n we already explained previously how can we managed it with only $O(log(n))$ bits. So the only thing we need to ensure is that the array S is not taking more than $O(log(n))$ space:

Lemma 1 *through all the run of the algorithm S use bits at most as the depth of the biggest full subtree of F +1 .*

By the lemma we conclude S is using at most $log(n)$ bits, as the size of the biggest full subtree of F is smaller than n and it's depth is $log(n)$.

Proof of lemma by induction on the size of the subtree the algorithm check in the *loop*. If subtree is in size of one (literal) we don't use S. Assume that for any subtree F' we checked in *loop* of size $k, k < n$ the if F' was full subtree then S grew at most to the size of F''s height and otherwise it grew to size of the biggest full subtree of F' +1 . Let be F' be a subtree of size n. If F' is full tree with length l, so the calculation on it's left subtree (which is also full subtree) will take at most $l-1$ bits in S (by induction assumption), after we go back to F' and before checking the right subtree, S will shrink to only one value. Then we will use additional at most $l-1$ bits for the right subtree of F', by the induction assumption, with total of l bits. Otherwise F' is not a full tree denote by l the height of the biggest full subtree of F'. To calculate the first subtree of F' by IA we will at most $l+1$ bits in S, but after coming back to F', S' will shrink back to size 1. It could not be the case that the biggest full subtree of F' is direct child of it in the second subtree calculation, as we scan F' first by the biggest child and this will lead to the fact that F' is full by it own, which is not the case we are in. Therefore to calculate the second subtree we will need, by IA, $l+1$ which will shrink to size two after coming back to second subtree of F' (one saved for the first subtree and the second bit for the second subtree). Then we will go back to F' and finish.

b.

SAT is a special case of boolean formula, so as we showed in 1, iff $x \in SAT$ there is a witness for it, a interpretation for it a with:

$$(x, a) \in FVAL$$

Because the witness tape is just like a normal tape we can run the algorithm from section a, and determine $x \in SAT$ iff $\exists a$ s.t. $(x, a) \in FVAL$.

c.

Claim 1.1 *G is not bipartite graph $\iff$ there is a cycle of odd length in G*

We will show that $BIPARTITE \in CO-NL$ and by theorem we showed in class ($NL = CO-NL$) we will conclude $BIPARTITE \in NL$. Let us show $\overline{BIPARTITE} \in NL$ by describe M non deterministic TM which decide $\overline{BIPARTITE}$.
M will receive as an witness the odd cycle in G (by our claim we know there exist iff $G \in \overline{BIPARTITE}$) and just check that this cycle exists in G.
M will accept $\iff$ there is such witness $\iff$ $G \in \overline{BIPARTITE}$.

Proof of claim *($\Leftarrow$) Let $C = \{v_1, v_2, \ldots, v_{2n+1}\}$ be a cycle of odd length, by contradiction assume $G = (V \cup E)$ is bipartite with $V = L \cup R$ division of V to two disjoint sets as promised. WLOG assume $v_1 \in L$ so $v_2 \in R$, $v_3 \in L$ and so on (there is edge between those vertices so they have to be*

in different sets), we conclude all the odd vertices are in L but there is edge between v_{2n+1} to v_1 and we arrive to contradiction.
($\Rightarrow$) Let be ■

Question 5

a.

Let be f a one-directional function, and we will show $P = NP$.
Let M_f be the polynomial TM which compute f.
Let us define TM $M(y, x)$,

- given (y, x)
- M runs $M_f(x)$ and get y'
- M accepts iff $y' = y$

So M is polynomial. We will define now L':

$$L' = \{(y, x, 1^n) : \exists z \ s.t \quad M(y, x \cdot z) = 1 \wedge |x \cdot z| = n\}$$

We will show $L' \in NP \setminus P$:

- $L' \in NP$: We will show a non deterministic TM M' which will decide L'. Given $(y, x, 1^n)$, M' will will guess $z \in \{0,1\}^{n-|x|}$ and run $M(y, x \cdot z)$ and answer the same. We will have:

$$(y, x, 1^n) \in L' \iff \exists z \ s.t \quad M(y, x \cdot z) = 1 \wedge |x \cdot z| = n \iff \\ \textit{there exists accepting computation for } M'$$

- $L' \notin P$: Otherwise by contradiction $L' \in P$, there exists TM M_p s.t

$$(y, x, 1^n) \in L' \iff M_p(y, x, 1^n) = 1 \iff \exists z \ s.t \quad M(y, x \cdot z) = 1 \wedge |x \cdot z| = n$$

  So we can build the following algorithm $\mathcal{A}$ which reverse f, like that, given y and $1^{|x|}$, $\mathcal{A}$ will set the first bit of x to 1 if the run of $M_p(y, 1, 1^{|x|})$ accepts, and 0 otherwise. Continue by running $M_p(y, x_1 1, 1^{|x|})$ to determine the second bit of x and after total $|x|$ runs of M_p we will find x, and hence $\mathcal{A}$ is polynomial in $(y, 1^{|x|})$. We got:

$$\Pr_{x^R \in \{0,1\}^*, f(x)=y}[\mathcal{A}(y) = x' \quad s.t. \quad f(x') = y] = 1 > \frac{1}{n}$$

  for all $n > 1$, and contradiction that f is one-directional function.

b.

Assume F is running in n^c, $c > 2$, for input in size n, we will define new function F' which will run in time n^2. Given x, F' will run F on the first $|x|^{\frac{2}{c}}$ bits of x (deleting all the bits after those first bits) and return. Clearly the run time of F' is $n^{\frac{2}{c}c} = n^2$. If by contradiction F' is not one-directional, there is algorithm $\mathcal{A}$ s.t.

$$\Pr_{x^R \in \{0,1\}^*, f(x)=y}[\mathcal{A}(y) = x' \quad s.t. \quad F'(x') = y] > \epsilon(n) \tag{3}$$

For any negligible function $\epsilon(n)$. So we can describe new algorithm $\mathcal{A}'$ which will contradict the face F is one-directional. Given $(y, 1^{|x|})$, for some $x \in \{0,1\}^*$ which $F(x) = y$, $\mathcal{A}'$ will run $\mathcal{A}$ on $(y, 1^{|x|^{\frac{c}{2}}})$ which will return x' s.t $F'(x') = y$ with high probability [as described in 3], $\mathcal{A}'$ return the first $|x'|^{\frac{2}{c}}$ of x', denote the output by x'', the size of output will be:

$$|x''| = |x'|^{\frac{2}{c}} = |x|^{\frac{c}{2}\frac{2}{c}} = |x|$$

By the definition of F', x'', the first $|x'|^{\frac{2}{c}}$ bits of x', are fulfilling $F(x'') = y$, and we arrive to contradiction that F is one directional.