Complexity - Exercise 1

Oren Roth

November 28, 2017

Question 1

($\Leftarrow$) Let $L^* \in NP-C \cap CO-NP$ and let us show $CO-NP = NP$.

Claim 0.1

$$L \in NP \Rightarrow \bar{L} \in NP$$

Proof of claim *$L \in NP$ and $L^* \in NP-C$ so there a polynomial reduction function f s.t.:*

$$L \leq_p L^*$$

Therefore:

$$x \in L \iff f(x) \in L^*$$

By the definition of complement languages:

$$x \in \bar{L} \iff x \notin L \iff f(x) \notin L^* \iff f(x) \in \bar{L^*}$$

So we conclude that f is a polynomial reduction for:

$$\bar{L} \leq_p \bar{L^*}$$

And we know $L^* \in CO-NP$ so $\bar{L^*} \in NP$ and thus $\bar{L} \in NP$ ■

After we showed the proof of the claim we will use it to show $CO-NP = NP$. Let $L \in NP$, by the claim:

$$\bar{L} \in NP \Rightarrow L \in CO-NP$$

Let $L \in CO-NP$, so we know $\bar{L} \in NP$ and by the claim:

$$L \in NP$$

We conclude $NP = CO-NP$.

($\Rightarrow$) Let $NP = CO-NP$, we know $SAT \in NP-C$ so,

$$SAT \in NP \Rightarrow SAT \in CO-NP$$

and we showed there is language in $NP-C \cap CO-NP$

Question 2

1. Define:

$$Id : \Sigma^* \to \Sigma^*$$
$$\forall x \in \Sigma^*, \quad Id(x) = x$$

Let $A \subseteq \Sigma^*$ then:

$$x \in A \iff Id(x) = x \in A$$

And thus Id is reduction from A to A. Clearly Id runs $O(1)$.

2. It's false.
We know $L_{halt} \in NP - Hard \setminus NP$ so for any $L \in NP$:

$$L \leq_p L_{halt}$$

but it if:

$$L_{halt} \leq_p L$$

then $L_{halt} \in NP$ and contradiction. And hence,

$$L_{halt} \not\leq_p L$$

3. It's false.
$SAT, SAT - 3 \in NP - Complete$ and thus:

$$SAT \leq_p SAT - 3$$
$$SAT - 3 \leq_p SAT$$

But $SAT \neq SAT - 3$.

4. Let $A \leq_p B$ by function f, hence:

$$x \in A \iff f(x) \in B$$

By the definition of complement languages:

$$x \in \bar{A} \iff x \notin A \iff f(x) \notin B \iff f(x) \in \bar{B}$$

So we conclude that f is a polynomial reduction for:

$$\bar{A} \leq_p \bar{B}$$

Question 3

Claim 0.2 *For any $\phi = (x_1 \vee x_2 \vee \ldots \vee x_k)$ with $k \geq 4$ there is CNF ϕ' with 2 clauses in length smaller than k each, defined over $x_1, \ldots x_k$ and y(new variable). With the following property:*

$$\phi(X) = true \iff \exists y \ s.t \ \phi'(X, y) = true$$

Proof of claim *Let $\phi = (x_1 \vee x_2 \vee \ldots \vee x_k)$ with $k \geq 4$, we define:*

$$\phi' = (x_1 \vee x_2 \vee \ldots \vee x_{k-2} \vee y) \wedge (x_{k-1} \vee x_k \vee \neg y)$$

If there is X assignment for $x_1, \ldots, x_k$ s.t. $\phi(X) = true$, if one of x_{k-1}, x_k got true we set $y = true$ and $\phi'(X, y) = true$ (the first clause is true due to y and the second due to x_{k-1}, x_k). And otherwise if both x_{k-1}, x_k are not true it has to be the case that one of $x_1, \ldots x_{k-2}$ literals got true and setting $y = false$ gain $\phi'(X, y) = true$. On the other hand if $\phi'(X, y) = true$ for some y, if $y = true$ than it has to be the case than one of x_{k-1}, x_k are true and thus $\phi(X) = true$ and else if $y = false$ it has to be the case than one of $x_1, \ldots x_{k-2}$ literals got true and still $\phi(X) = true$. ■

Claim 0.3 *For any $\phi = \bigwedge_{i=1}^{n} C_i$, an CNF for SAT there is CNF $\phi' = \bigwedge_{i=1}^{m} C'_i$ for $3 - SAT$ with $poly(n)$ clauses on the same variables. ϕ, ϕ' will agree between them. Moreover ϕ' can be build from ϕ in polynomial time.*

Proof of claim *Let $\phi = \bigwedge_{i=1}^{n} C_i$, an CNF for SAT, define $k = max\{|C_i|\}$ we will show by induction on k that we can create ϕ'.*
Basis:

1. ***k=1:*** *instead of $C_i = x_1$ we will create:*

$$C'_i = (x_1 \vee y_i \vee z_i) \wedge (x_1 \vee \neg y_i \vee z_i) \wedge (x_1 \vee y_i \vee \neg z_i) \wedge (x_1 \vee \neg y_i \vee \neg z_i)$$

   With two new variables (y_i, z_i), and no matter what is the assignment of y_i, z_i it will be one clause in C'_i that x_1 will need to have true. On the other side if x_1 is true than all the clauses in C'_i are true.

2. ***k=2:*** *Clauses of length 1 we will handle as before, otherwise, instead of $C_i = (x_1 \vee x_2)$ we will create:*

$$C'_i = (x_1 \vee x_2 \vee y_i) \wedge (x_1 \vee x_2 \vee \neg y_i)$$

   With new variables (y_i), again no matter what is the assignment of y_i it will be one clause in C'_i that x_1, x_2 will need to have true. On the other side if x_1, x_2 is true than all the clauses in C'_i are true.

3. ***k=3:*** *Clauses of length < 3 we will handle as before, otherwise, we will keep $C'_i = C_i$ as it has 3 literals in each clause.*

Assume that for any k with $k < n$ we can define ϕ' and let be ϕ with $k = max\{|C_i|\} = n \geq 4$. For any C_i with length n by the previous lemma we know we can create two clauses with length $n-1$ that has the same truth values and know we have new formula with all clauses of length $< n$ and by the assumption we can handle those.
Notice that for each clause with m literals we create at most m new clauses, and so this process is polynomial in n. ■

It's clear $3-SAT \in NP$ as given ϕ an assignment for ϕ can be the proof for showing whereas $\phi \in 3-SAT$ (As there is a valid assignment for ϕ iff $\phi \in 3-SAT$). To show $3-SAT \in NP-hard$ we will just show polynomial reduction between SAT to $3-SAT$, because $SAT \in NP-Complete$ it will admit immediately that $3-SAT \in NP-hard$.
Given, $\phi = \bigwedge_{i=1}^{n} C_i$, an CNF for SAT we will build in polynomial time ϕ' for $3-SAT$ by the lemma, and we got $\phi \in SAT \iff \phi' \in 3-SAT$.

Question 4

We will use an array Arr with size m.

```
   H(m) :
1  Arr[1] ← 1
2  for M=2 to m do
3      i=1
4      if i < loglog(m) then
5          for x ∈ {0,1}^{log(M)} do
6              run M_i(x) for i|x|^i steps and put the result in res1 (e.g res1 = true iff M_i(x) stop
                 and accept after at most i|x|^i steps)
7              l ← the index of the rightmost '0' in x
8              x_start ← x[0, l]
9              x_end ← x[l+1, |x|]
10             if |x_end| ≠ |x_start|^{Arr[|x_start|]} then
11                 res2 ← false
12             end
13             else if x_start is not CNF clause then
14                 res2 ← false
15             end
16             else
17                 brute force to find satisfing assignment for x_start and set res2 = true iff one
                     found.
18             end
19             if res1 ≠ res2 then
20                 i ← i + 1
21                 goto 4
22             end
23         end
24         Arr[M] ← i
25     else
26         Arr[M] ← loglog(M)
27     end
28 end
29 return Arr[m]
```

Explanation for the run time:

- The loop in line 2 runs in linear time in m.

- We go back to line 4 at most $loglog(m)$ times.
- The loop in line 5 runs at most $2^{log(M)} < 2^{log(m)} = m$ times.
- Line 6 runs in $i|x|^i < loglog(m)log(m)^{loglog(m)} = O(m)$ steps*.
- Line 7-12 run in polynomial time in x which is polynomial in m.
- The CNF clause in Line 13 has at most $|x| = log(M) < log(m)$ variables and thus need to check at most $2^{log(m)} = m$ assignments.

[*] $$\lim_{m\to\infty} \frac{\log(\log(m)) \log^{\log(\log(m))}(\log(m))}{m} = 0$$

Summary - Each of the lines run in polynomial time in m, and because polynom of polynom is still polynom also finite set of cascading loops are still polynom in m.