CS5110 - Assignment-2

C. Akshay Santoshi
CS21BTECH11012

1. $A \leq_L B$ using reduction f. w is an instance of A. Upper bound on $|f(w)|$?

Let us take $|w| = n$.

$DSPACE(O(f(n))) \subseteq DTIME(2^{O(f(n))})$

$DSPACE(O(\log n)) \subseteq DTIME(2^{O(\log n)})$
$= n^k$

For the logspace reduction, time taken would be polynomial in n. At each time step, we write one symbol, so $|f(w)|$ could be as bad as n^k.

Upperbound on $|f(w)|$ would be $O(|w|^k)$.

2. $A_{NFA} = \{<N,w> \mid N$ is an NFA that accepts the string $w\}$.

~~A~~ $A_{NFA} \in NL$

$N = (Q, \delta, q_0, F, \Sigma)$
$q_0 = s$

For each character of w, we can nondeterministically guess a new state from the current state, check whether it is a valid transition. If yes, go to that state. Continue this till all characters of w are iterated over. If finally, we end up in an accepting state, we output YES, otherwise NO. Note that at any instance we are only storing the current ~~state~~ node we are in. Therefore $A_{NFA} \in NL$.

$PATH \leq_L A_{NFA}$

$PATH = \{<G,s,t> \mid$ There is a path from s to t in $G\}$.
$G = (V, E)$.

We construct NFA, N as follows:
We consider $\Sigma = \phi$.
$Q = V$, $q_0 = s$, $F = \{t\}$.

If we have an edge from a to b in G, then transition in N would look like $a \xrightarrow[\varepsilon]{} b$.

We take $w = \varepsilon$.

This construction would take only logspace. We don't have to remember all at once. We can output Q as vertices of graph by outputting single node each time.

Similarly for the rest of q_0, F, Σ and δ. At a step, we only write a vertex or an edge. This takes only logspace.

$w = \varepsilon$.

Now N accepts w iff there is a path from s to t in G.

($\Rightarrow$) If N accepts w, then this means there are 0 or more nodes (states) between s and t, so that using ε transitions, you can end up at 't' from 's'.
These transitions exist in NFA because there is a path ~~to~~ from s to t. (~~For~~ From construction)

($\Leftarrow$) If there is a path from s to t in G, since $w=\varepsilon$, N can non-deterministically go through the states (each selected at a time using the nodes present in the ~~graph~~ path) and reach 't' through ε transitions.

$\therefore$ A_{NFA} is NL-complete.

3) 2-SAT is NL-complete.

<u>2-SAT $\in$ NL</u>

NL = co-NL. So, we will show that $\overline{\text{2-SAT}} \in$ NL.

2-SAT is of the form $(x_1 \vee x_2) \wedge (x_3 \vee \bar{x}_1) \wedge \dots \wedge (\quad)$

We construct a graph with nodes $x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n$.

Edges are constructed as follows:

For every clause $(a \vee b)$, $a \vee b$ becomes true when atleast one of a or b is true.

If a is false, b has to be true $\equiv \bar{a} \to b$

If b is true, a has to be true $= \bar{b} \to a$.

So for every clause $(a \vee b)$, we have edges $\bar{a} \to b$ and $\bar{b} \to a$.

If for a variable 'a', you get a chain like this: $\bar{a} \to b \to \bar{c} \to \dots \to a$.
It forces $\bar{a}$ to be False (a to be True), since both $\bar{a}$ and a appear in same chain.

Another possibility: $a \to b \to c \to \dots \to \bar{a}$.
This forces a to be False.

So ϕ would be unsatisfiable iff there is a cycle in G which has both x and $\bar{x}$.
This can be solved in a similar way that we used to approach PATH∈NL.
At any instant, we just store a vertex and a counter.

for i=1 to n.

PATH$(G, x_i, \bar{x}_i)$
PATH$(G, \bar{x}_i, x_i)$

If both are present then we say that ϕ is unsatisfiable.

For the next vertex while choosing, we non-deterministically guess a vertex (check if it has an edge to it) and go to that vertex.

$\therefore \overline{\text{2-SAT}} \in \text{NL}$.

Since NL = co-NL, 2-SAT ∈ NL.

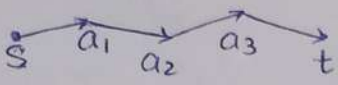
$\underline{\text{PATH}^C \leq_L \text{2-SAT}}$

PATH ∈ NL-complete. Since NL=co-NL, $\text{PATH}^C \in \text{NL}$.
We'll show reduction from $\text{PATH}^C \leq_L$ 2-SAT.

$\text{PATH}^C = \{\langle G,s,t\rangle \mid$ In G, there is no path from s to t $\}$.

Given $G=(V,E)$ and s,t, we construct 2-SAT, ϕ as follows:
For every edge $(u,v\ (u \to v)) \in E$, we have a clause $(\bar{u} \vee v)$
Additionally, we also have clauses $(s \vee s)$ and $(\bar{t} \vee \bar{t})$.
Now, we show that there is no path from s to t iff ϕ has satisfying assignment.

($\Leftarrow$) If ϕ has satisfying assignment then no path from s to t in G.
We prove contrapositive, i.e, if there is a path from s to t in G, then there is no ~~#~~ satisfying assignment for ϕ.



So our construction of ϕ for this would be:

$(\bar{s} \vee a_1) \wedge (\bar{a}_1 \vee a_2) \wedge (\bar{a}_2 \vee a_3) \wedge (\bar{a}_3 \vee t)$

Assume it to be satisfiable.

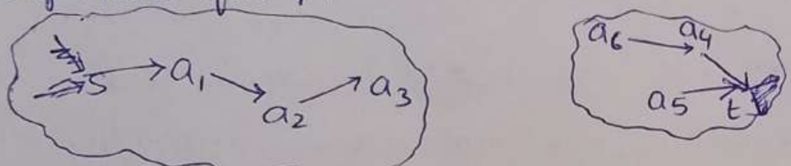
For $(s \vee s)$ to be T, clearly s has to be True. So a_1 has to be T. Which forces a_2, a_3 and t to be true.

But $(\bar{t} \vee \bar{t})$ would be F. Contradiction.

So there is no satisfying assignment for ϕ if there is a path from s to t.

$(\Longrightarrow)$

If there is no path from s to t then there is a satisfying assignment for ϕ.



Since there is no path from s to t, we can make partitions like this. There may exist additional partitions also.

Our ϕ would ~~lik~~ look like this:

$(\bar{s} \vee a_1) \wedge (\bar{a}_1 \vee a_2) \wedge (\bar{a}_2 \vee a_3) \;\ldots\; \wedge (\bar{a}_6 \wedge a_4) \wedge (\bar{a}_4 \wedge t) \wedge (\bar{a}_5 \wedge t) \ldots$

$\wedge (s \vee s) \wedge (\bar{t} \vee \bar{t})$.

We can assign 's' to be True, 't' to be False. All nodes which belong to the partition containing 's' to be True and all nodes which belong to the partition containing 't' to be False.

Therefore we get a satisfying assignment for ϕ.

$\therefore$ 2-SAT $\in$ NL-complete.

4. $\text{LADDER}_{\text{DFA}} = \{\langle M, s, t\rangle \mid$ M is a DFA and L(M) contains a ladder of strings, starting with 's' and ending with 't'$\}$

Given M, s, t. First check whether $|s| = |t|$. Reject if not.

We can construct a graph G as follows:
Vertices would be strings of length $|s|$ from english alphabet.
Now, edges would be between vertices (which are indexed by strings) which differ by only one letter and both of those strings $\in L(M)$.
Total no of vertices are $|\Sigma|^{|s|}$.

$\langle M, s, t \rangle \in LADDER_{DFA}$ iff there is a path from s to t in G.
We start at 's'. We non-deterministically select the next vertex ~~by~~ which differ by only one letter. We check whether M accepts w. At any point of time, we only store the current vertex we are in.
We also maintain a counter used as timer.
If counter reaches $|\Sigma|^{|s|}$ and machine M hasn't accepted till now, we can output REJECT. Else, we ACCEPT. (This is because if there was a path from s to t, the length of the path would be atmost $|\Sigma|^{|s|}$ if we remove loops).
So, this uses NPSPACE.
Since PSPACE = NPSPACE, $LADDER_{DFA}$ is in PSPACE.

5) If $A \in P$, then $P^A = P$.
P^A is the set of all languages that can be decided by a poly time det. T.M with oracle access to A.

$\underline{P \subseteq P^A}$

This is trivial in the sense that A only adds more power. Even without accessing A also, we can decide whether a language is in P or not. So, any language in P can be decided by P^A machine by choosing not to (or choosing to) use the oracle calls.

$\underline{P^A \subseteq P}$

Suppose $L \in P^A$.
Then we can have a poly-time det T.M without oracle access

by replacing each oracle call to A with a polytime computable algorithm that decides A, since given that $A \in P$. Since $A \in P$, oracle calls are poly time, these computations require product of poly times, which is also poly time.

$\therefore$ We can now decide L in poly time without oracle access.

So $L \in P$

$\therefore \underline{\underline{P^A = P}}$

6) STRONGLY-CONNECTED $\in$ NL

For all ordered pairs (x, y), we check nondeterministically for a path from x to y.

We know PATH $\in$ NL. At any instance, we store only the vertex and the counter.

If for any pair (x', y'), we fail to non-deterministically get a path from x' to y', we reject.

If G is strongly connected, then every pair would have a sequence of guesses which lead to a valid path. If not, then there is no path for atleast one pair and hence not strongly connected.

PATH $\leq_L$ STRONGLY-CONNECTED:

PATH = $\{ \langle G, s, t \rangle \mid$ In G, there is a path from s to t $\}$

~~We~~ Given $G = (V, E)$, we construct G' with $V' = V$ and E' such that E' contains E and also some additional edges for every $v \in V$, we construct edges $(t \to v)$ and $(v \to s)$.

This is a logspace reduction, since we deal with atmost two vertices at a time to construct the edges.

G' is a strongly connected graph iff G has a path from 's' to 't'.

$(\Leftarrow)$ ~~If G~~

If G has a path from 's' to 't', then G' is strongly connected. For any two vertices 'u' and 'v', we have an edge (u, s) through construction, path from s to t (since G has a path

from 's' to 't'), edge (t, v) (through construction).
So we have a path from u to v for every such pair.
$\therefore$ G' is strongly connected.

($\Rightarrow$)
If G' is strongly connected, then there is a path from s to t in G.
This is because, the new edges we constructed in G' have no outgoing edges from 's' and no incoming edges from 't'.
So path from 's' to 't' exists only when there is such a path in G. Hence proved.

$\therefore$ STRONGLY-CONNECTED is NL-complete.

7) $L \subseteq \{0,1\}^*$ $\quad L_f \subseteq \{0,1,\#\}^*$ $\quad L_f := \{x\#^{f(|x|)} \mid x \in L\}$.

(a) If $L \in \text{DTIME}(f(n))$. Then show that $L_f \in \text{DTIME}(O(n))$.

Sol:- First, verify whether the input w is of the form $x\#^m$ where $x \in \{0,1\}^*$ and ~~x≥~~ $m \geq 0$. If not, reject immediately.
Check if $x \in L$ by using the DTIME(f(n))-algo for L. Since $L \in \text{DTIME}(f(n))$, this step takes $O(f(|x|))$ time.
Compute $f(|x|)$ in $O(f(|x|))$ time and check whether the no. of # symbols is exactly $f(|x|)$.
If it matches, ACCEPT. Otherwise REJECT.

(b) If $f(n)$ is poly. function, then $L \in P \Leftrightarrow L_f \in P$.

($\Rightarrow$)
If $L \in P$, then using the same reason as before, we can decide L and compute $f(n)$ in poly-time. $\therefore L_f \in P$.

($\Leftarrow$)
If $L_f \in P$.
Suppose given input x, we want to decide whether it is in L or not. We can compute $f(|x|)$ in poly-time (given).
Pad x with $f(|x|)$ many #'s.

Now provide this padded string as input to the poly-time decider, M of L_f. Output whatever M outputs.
This is because $x\#^{f(|x|)} \in L_f$ iff $x \in L$.

~~3~~ (c) $P \neq DSPACE(O(n))$.

Assume $P = DSPACE(O(n))$.

Let $L \in DSPACE(O(n^2))$. We take $f(n) = n^2$.

So, we get $L_f \in DSPACE(O(n))$.

From assumption, $DSPACE(O(n)) = P$, so $L_f \in P$.

From problem (b), if $L_f \in P$, then $L \in P$. So, $L \in DSPACE(O(n))$.

We got that if $L \in DSPACE(O(n^2))$, then $L \in DSPACE(O(n))$.

This contradicts Space Hierarchy Theorem since it states that there are languages in $O(n^2)$ but not in $DSPACE(O(n))$.

$\therefore P \neq DSPACE(O(n))$

(d) $NEXP = \bigcup_k NTIME(2^{n^k})$.

If $P = NP$, then $EXP = NEXP$.

Assume $P = NP$

Take a language $L \in NTIME(2^{n^k})$.

Define $f(n) = 2^{n^k}$, so that L_f is the padding extension of L.

Since $L \in NTIME(2^{n^k})$, we have $L_f \in NP$.

By $P = NP$, we have $L_f \in P$.

So we have a det. machine for L_f and $f(n) = 2^{n^k}$, $L \in DTIME(2^{O(n^k)})$.

~~Theref~~ Therefore, we got that if $L \in NTIME(2^{n^k})$, then $L \in DTIME(2^{O(n^k)})$.

$\therefore NEXP \subseteq EXP$.

We already know $EXP \subseteq NEXP$, since any L which can be determined by det. T.M can be determined by a non-det. T.M. as well.

$\therefore NEXP = EXP$.

8. If $\Sigma_K = \Pi_K$, the P.H collapses to Σ_K.

Let $L \in \Sigma_{K+1}$.

$L = \{x \mid \exists y_1 \underbrace{\forall y_2 \exists y_3 \ldots Q_{K+1} y_{K+1}}\}$ $|y_1|, |y_2|, \ldots, |y_{K+1}| = poly(|x|)$ and $M(x, y_1, y_2, \ldots, y_{K+1}) = 1$.

This part is of the form Π_K.

$M'(\cancel{y_1}(x, y_1), y_2, y_3, \ldots, y_{K+1}) = 1$ for $|y_i| = poly(|x|)$ $\forall i$.

Since given that $\Sigma_K = \Pi_K$, that part of the expression can be replaced with

$L = \{x \mid \exists y_1 \exists z_1 \forall z_2 \forall z_3 \ldots Q_K z_K\}$ $|z_i| = poly|x|$ $\forall i$

$M''(\cancel{y_1} x, y_1, z_1, z_2, \ldots, z_K) = 1$.

$= \{x \mid \underbrace{\exists (y_1, z_1) \forall z_2 \exists z_3, \ldots Q_K z_K}\}$. $|z_i| = poly|x|$ $\forall i$.

This expression is of the form which belongs to Σ_K.

$\therefore L \in \Sigma_K$.

We got that if $L \in \Sigma_{K+1}$, then $L \in \Sigma_K$. $\therefore \Sigma_{K+1} = \Sigma_K = \Pi_K$.

Now, if we take $L' \in \Pi_{K+1}$.

$L' = \{x \mid \forall y_1 \underbrace{\exists y_2 \ldots Q_{K+1} y_{K+1}}\}$ $|y_i| = poly(|x|)$ $\forall i$ $N(x, y_1, y_2, \ldots, y_{K+1}) = 1$

$N'((x, y_1), y_2, \ldots, y_{K+1}) = 1$

This is of the form Σ_K.

Since given that $\Sigma_K = \Pi_K$.

$L' = \{x \mid \forall y_1 \forall z_1 \exists z_2 \ldots Q_K z_K\}$ $|z_i| = poly(|x|)$ $\forall i$ $N''(x, y_1, z_1, z_2, \ldots, z_K) = 1$.

$= \{x \mid \underbrace{\forall (y_1, z_1) \exists z_2, \ldots Q_K z_K}\}$ $|z_i| = poly(|x|)$ $\forall i$

This expression is of the form which belongs to Π_K.

$\therefore L' \in \Pi_K$.

We got that if $L' \in \Pi_{K+1}$, then $L' \in \Pi_K$.

$\therefore \Pi_{K+1} = \Pi_K = \Sigma_K = \Sigma_{K+1}$.

By induction, it can be extended to show that the whole P.H collapses down to Σ_K.

9) Count no. of functions that are both monotone and symmetric.

f is monotone if flipping any input bit from 0 to 1 never decreases the output of the function.

f is said to be symmetric if its output depends only on the no. of 1s in the input and not on the specific arrangement or order of the 0s and 1s in the input.

Let us define h such that,

h(0) means output of f when no. of 1's ~~in the~~ is 0 in the input.

h(1) means output of f when no. of 1's is 1 in input.

From monotonicity, $h(0) \leq h(1) \leq h(2) \ldots \leq h(n)$.

If we fix h(0) as 1, then $h(1) = \ldots = h(n) = 1$.

If we take 'i' to be the smallest no. for which $h(i)=1$, then $h(j) = 1 \ \forall j \geq i$.

'i' can take values $0, 1, \ldots, n$. — $(n+1)$ functions

Additionally f can be constant function which outputs 0 irrespective of the input.

So, in total there are $n+2$ such functions.