CS6014 : Advanced Theory of Computation — Karthik Abinav S

Problem Set #6 — Collaborators : Saikrishna

1. (10 points) Oracle Queries.
   (a) Argue that $\mathsf{P}^{\mathsf{P}} = \mathsf{P}$. Why does not the same argument work for NP and give $\mathsf{NP}^{\mathsf{NP}} = \mathsf{NP}$?
   (b) If $\mathsf{NP} = \mathsf{coNP}$, argue that $\mathsf{PH} = \Sigma_1^p$.

Solution:

(a) $\mathsf{P}^{\mathsf{P}} \subseteq P$
Let $L \in \mathsf{P}^{\mathsf{P}}$. There exists an oracle TM N such that, it runs in polynomial time and has access to a language in P as an oracle. This can be simulated by deterministic polynomial time machine M without the oracle as follows :

M runs on the input in the same way as N. Except whenever N makes query to the oracle to check if $x \in K$ where K is the oracle, M runs the machine corresponding to the oracle language on x and gets the answer. It then proceeds in the similar way as the machine N. Since the oracle is a language in P, the simulation of the query to oracle takes deterministic polynomial time. Hence , M runs in polynomial time.

$\mathsf{P} \subseteq \mathsf{P}^{\mathsf{P}}$
This containment is trivially true, since the machine accepting P need not use the oracle and hence belongs to P^{P}

From above two containments, $\mathsf{P}^{\mathsf{P}} = \mathsf{P}$

The above argument wont work for NP because, when the oracle gives the answer as no, the same thing cannot be determined by simulating a NP machine in polynomial time. Since , the time for a non deterministic machine is defined as the length of the shortest accepting path. Hence,the rejecting paths in the non deterministic tree may not be of polynomial length.

(b) This can be shown using induction on i of Σ_i^P .
Base case. $i = 2$.
we have to show $\mathsf{NP}^{\mathsf{NP}} = \mathsf{NP}$. The reverse containment is trivially true. To show the forward containment, we can simulate the queries on the oracle by simulating the two non deterministic machines corresponding to L(say M) and

$\overline{L}$(say N) . There exists a non deterministic polynomial time machine for $\overline{L}$ because NP = coNP.
Whenever the original machine made a query to oracle, simulate both the non deterministic machine in time sharing fashion. If M accepts, interpret as a yes answer by the oracle and continue calculation as the original machine. If N accepts, interpret as a no answer by the oracle and continue calculation as the original machine. Since, both M and N accepts in poylnomial time, this new machine also runs in polynomial time. Hence, $\Sigma_2^P = \Sigma_1^P$.

Let for some $k > 2$, $\Sigma_k^P = \Sigma_1^P$.
Σ_{k+1}^P is language which can be accepted in NP time given an oracle to Σ_k^P. Since, from induction hypothesis, $\Sigma_k^P = \Sigma_1^P$, this can be accepted by a machine in NP time which has oracle access to NP. Similar to the above case, let M be a machine which accepts the language corresponding to $L \in$ NP and $\overline{L} \in$ coNP . Now, the oracle query can be replaced by simulating both the machines in a time sharing manner. If M accepts, interpret as a yes answer from oracle. If N accepts, interpret as a no answer from oracle. Then continue computation normally as the oracle machine from there. Since, both M and N accepts in polynomial time, hence the entire simulation can be done in NP time. Therefore, $\Sigma_{k+1}^P = \Sigma_1^P$.

2. (10 points) (a) Reading Assignment : Read the proof (Section 3.4, Theorem 3.21, Page 93) of the claim : $\mathsf{DTIME}(2^{O(s(n))} \subseteq ASPACE(s(n))$. Determine an upper bound on the number of children for any universal configuration in the alternating Turing machine produced in the construction.

   (b) Conclude that $AL = \mathsf{P}$. Show that all CFLs are in P by giving an alternating Turing machine running in log space for checking membership. (Assume that the CFL is given at the input in the CNF form.).

Solution:

(a) From the construction in the proof a universal configuration node has atmost 3 existential children nodes and a existential node has atmost $|\Gamma^3|$ universal children nodes. where Γ is the tape alphabet after tape reduction.

(b) From the theorem, $\mathsf{DTIME}(2^{O(s(n))}) = ASPACE(s(n))$.
$AL = \bigcup ASPACE(\log(n))) = \bigcup DTIME(2^{O(\log(n))}) = \bigcup DTIME(2^{c\log(n)}) = \bigcup DTIME(n^c) = \mathsf{P}$

3. (10 points) Dene the language:

   ShortestPath $= (G, k, s, t)|$ the shortest path from s to t in G has length k

(a) (5 points) Prove that ShortestPath is in NL.

(b) (5 points) Prove that ShortestPath is in L if and only if L = NL.

Solution:

(a) We can show ShortestPath $\in$ NL by showing tow things. First, there exists a path from s to t of length exactly k. Second, there doesnt exist of path from s to t of length 1,2,...k-1 .

To show the first part we will use the Reach algorithm with a slight modifications. We will use an extra $\log(n)$ bits to keep a ounter of the length of the path visited so far. Since , the length of the path can atmost be n(Otherwise, there will be a loop from Pigeon Hole principle and the loop can be eliminated) hence $\log(n)$ bits will suffice.

Let us call this modified version of Reach as KReach. Clearly KReach $\in$ NL . From Immerman-Szelepsinyi Theorem, NL = coNL. Hence, there exists a NL algorithm for $\overline{\text{KReach}}$.

We will use this algorithm as a sub routine and call it with (G,s,t,1) , (G,s,t,2) ... (G,s,t,k-1) , reusing the same space across the calls. Hence, the second part can also be done in NL. Therefore, ShortestPath $\in$ NL.

(b) L = NL $\implies$ ShortestPath $\in$ L

This direction is trivial from part a. Since ShortestPath $\in$ NL and L = NL, hence, ShortestPath $\in$ L.

ShortestPath $\in$ L $\implies$ L = NL
Let ShortestPath $\in$ L. Then there exists a deterministic log space algorithm. We can use this a subroutine and solve Reach in L as follows.
Call the ShortestPath algorithm with (G,s,t,1) , (G,s,t,2) ... (G,s,t,n) . From the same argument as above, if there exits a path from s to t, then there exists a path of length atmost n from s to t. The number of bits required to represent the number 1,2..n will be atmost $\log(n)$ and the algorithm takes another $\log(n)$ sapce which can be reused across calls. Hence,Reach can be computed in deterministic logspace. Since, Reach is NL-Complete, hence, every language in NL can be reduced to L . Hence, L = NL.

4. (15 points) An undirected graph is bipartite if its nodes can be divided into two sets such that all edges go from a node in one set to a node in the other set. Show that a graph is bipartite if and only if it does not contain a cycle that contains an odd number

of nodes. Let

$$\textsc{Bipartite} = \{G \mid G \text{ is a bipartite graph } \}$$

Show that BIPARTITE is in NL. (Hint : Use Immerman-Szelepsinyi theorem !).

Solution:

(a) Bipartite graph $\implies$ graph contains no odd cycles.

This can be shown by considering the contrapositive of the statement. If a graph contains odd cycles, then it is not bipartite. Consider any odd cycle $c_1, c_2 ... c_{2n+1}$. To form a bipartite graph using these vertices, we have to select vertices in one set such that, no two of them are adjacent in the cycle. Selecting set of vertices and forming two such sets, will leave out one vertex, for any combination. Hence such a set of vertices can never be bipartite.

Graph contains no odd cycles $\implies$ Bipartite graph .

Divide the set of vertices in the graph into set A and $V \setminus A$.
Every vertex $v \in A \Leftrightarrow$ the shortest path from every vertex in A to v is of odd length.

For such a construction of set, if there is no odd cycle, the graph is bipartite. Suppose there is no odd cycle, and let v_1 and v_2 both in A be adjacent. Consider a vertex $v \in A$. the length of the closed cycle from v will be length of $v...v_1 + v_2...v + 1$. From the definition of set, this length is odd . (odd + odd + odd) . BUt our initial assumption was there is no odd cycle. Hence, there cannot be v_1 and v_2 in A such that they are adjacent. Hence graph is bipartite.

(b) Consider the language $\overline{BIPARTITE}$ which is the complement of $BIPARTITE$. We can show that $\overline{BIPARTITE} \in$ NL from the result in part(a). That is given a graph G, if there exists a odd cycle, then $G \in \overline{BIPARTITE}$. Since, we already know REACH $\in$ NL, we have to check if any of (G,v_1,v_1) ,(G,v_2,v_2) .. (G,v_n,v_n) is in REACH. This can also be done in log space because, we can reuse the same space of (G,v_1,v_1) for checking (G,v_2,v_2),...(G,v_n,v_n).
Hence, $\overline{BIPARTITE} \in$ NL and $BIPARTITE \in$ coNL.

From Immerman-Szelepsinyi Theorem, NL = coNL. Hence, $BIPARTITE \in$ NL.

5. (25 points) We define the product of two $n \times n$ Boolean matrices A and B as another $n \times n$ Boolean matrix C such that $C_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge B_{kj})$.

   (a) (5 points) Show that boolean matrix multiplication can be done in logarithmic space.

(b) (5 points) Using repeated squaring, argue that A^p can be computed in space $O(\log n \log p)$.

(c) (5 points) Show that if A is the adjacency matrix of a graph, then ($A^k_{ij} = 1$ if and only if there is a path of length at most k from the vertex i to vertex j and is 0 otherwise.

(d) (5 points) Use the above to give an alternative proof that $\mathsf{NL} \subseteq \mathsf{DSPACE}(\log^2 n)$. We originally proved it using Savitch's theorem.

Solution:

(a) To do boolean matrix multiplication, we need to store the indices i,j,k and the current value of $A_{ik} \wedge B_{kj}$. Since, each of i,j,k can be atmost n, each of them require $\log(n)$ bits to represent. To store the current value we need 1 bit. Hence, to calculate the value of C_{ij} we need $3\log(n) + 1$ bits. This space can be reused for the calculations of all the entries in the matrix C. Hence, the entire matrix multiplication of boolean values can be done in log space.

(b) The usual repeated squaring technique requires us to store the intermidiate matrices which will need O(n) space. Instead, we can slightly modify the algorithm so that, whenever we are calculating A^p_{ij}, we need to do a logical or over all k's which will need, $A^{p/2}_{ik}$ and $B^{p/2}_{ik}$. This can be calculated on the fly, by storing just the $A^{p/2}_{ik}$ and the $B^{p/2}_{ik}$. This will need a space of O($\log(p) * \log(n)$) space for finding the $A^{p/2}_{ik}$(Because to further find this entry we have to go all the way till A). This space can be reused for the other entries in A^p.

(c) This can be shown by using induction on length k.
Base:
The matrix A contains entries which trivially indicates the existence of a path of length 1 between pair of vertices. i.e. if a edge exists between the vertices, there is a path of length 1.
Inductive case:
Let, A^k ,$2 \leqslant k \leqslant n$ denote the entries such that if a path of length atmost k exists it is 1 , else 0.
$A^{k+1} = A^k * A$.
A^{k+1}_{ij} is 1 if A^k_{ir} is 1 and A_{rj} is 1. This can alternatively be interpreted as, if there is a path of atmost length k from vertex i to r and a path of length 1 from vertex r to j. And the length of this path from i to j will be atmost $k + 1$. Hence, all the 1 entries in A^{k+1} indicate the pair of vertices having a atmost $k + 1$ path between them.

(d) From part (c), A^n contains a 1 if there is a path of length atmost n. In a graph, if there exists a path between two vertices, then there exists a path of at most n length between those two vertices (Pigeon hole principle). Hence, if A^n_{ij} is 1 if and only if i is reachable from j. From part (b) , the matrix A^n can be

calculated in $O(\log(n) * \log(n))$ space.
For any language in NL, it can be reduced to $REACH$ under log space reductions. And $REACH$ can be solved in $DSPACE(\log^2(n))$ using the n th power of the adjancency matrix. Hence, $L \in \mathsf{NL} \implies L \in DSPACE(\log^2(n))$.
Hence, $\mathsf{NL} \subseteq DSPACE(\log^2(n))$.