CS452 HW9

Yuting Chen

8.6

By defination of PSPACE-hardness for a language L.

For all $A \in PSPACE$, $A \leq_p L$.

$SAT \in NP$, $SAT \in PSPACE$, since $NP \subseteq PSPACE$.

So, $SAT \leq_p L$.

Since SAT is NP-complete. ⇒ It's NP-hard.

Thus, $SAT \leq_p L$. L is NP-hard by transitivity of $\leq_p$.

Thus, We can get that any PSPACE-hard language is also NP-hard.

8.8

Let DFA M_1 with a states, DFA M_2 with b state.

Assume M_1, M_2 differ on some input iff they differ on some input of length at most ab.

Let w be a string of minimal length on which they differ. $|w| > ab$.

Only ab have possibile combinations of w.

So, By pigeonhole, there must be $j > i$ for which both M_1 and M_2 are at the same states after i or j symbol of w.

Delete $i+1$ through j of w.

Therefore, we can get a shorter string which takes M_1 and M_2 to same states w.

So, this shorter string on which they differ. This is contradiction.

Assume M_1, M_2 differ on some input iff they differ on some input of length at most 2^{a+b}.

Set a straightforward nondeterministic algorithm to decide if two DFAs differ as follow:

1. Check if the input does encode M_1 and M_2. If not, reject. Otherwise, Let $a = |M_1|$, $b = |M_2|$.
2. Store list of $S \subseteq Q_{M_1}$ as $Q(a)$ space, list of $T \subseteq Q_{M_2}$ as Q_b space, string length l as $a+b$ space.
3. Initialize S to be ϵ-closure of the initial state of M_1.
4. Initialize T to be ϵ-closure of the initial state of M_2.
5. Set $l = 0$. For $l = 0$ to 2^{a+b}.

I. If M_1 include a final state, but M_2 do not, or vice versa. Accept.

II. Nondeterministically choose a symbol $\sigma \in \Sigma$.

III. Update S to be the union $\bigcup_{s \in S} \delta_{M_1}(s, \sigma)$.
IV Update T to be the union $\bigcup_{t \in T} \delta_{M_2}(t, \sigma)$.
So, this is accepted iff there is a string of length at most 2^{a+b} on which M_1, M_2 differs.
Complement of EQ_{REX} lies in NPSPACE = PSPACE.
But PSPACE is closed under complement.
We can switch the accept and reject states to get a decider for the complement.
So, $EQ_{REX} \in$ PSPACE.

8.10

Set a algorithm decides whether player "X" has a winning strategy in instances of go-moku.
We can show that this algorithm runs in polynomial space.
Assume position p indicates which player is the next to move.
M = "on input <P>, where P is a position in generalized go-moku:
1. If "X" is next to move, it can win in this trun, then accept.
2. If "O" is next to move, it can win in this trun, then reject.
3. If "X is next to move, it cannot win in this trun, then for each free grid position p.
recursively call M on <P'> where P' is upated P with "X"'s marker on position p. and "O" is the next to move. If one or more of these accepts, accept. If none of these calls, reject.
4. If "O" is next to move, It cannot win in this trun, then for each free grid position p.
recursively call M on <P'> where P' is upated P with "O"'s marker on position p. and "X" is the next to move. If all of those calls accepts, accept. If one or more of these calls reject, reject.
The only space required by the algorithm is for storing the recursion stack.
Each level adds a single position.
So, the stack uses at most $O(n)$ space, at most n^2 levels.
So, the algorithm runs in $O(n^3)$.
Thus, we get a polynomial space complexity.
$\Rightarrow$ GS $\Rightarrow$ PSPACES.

8.11

Given if every NP-hard language is PSPACE-hard, then SAT is PSPACE-hard.
Consequently every PSPACE language is polynomial-time reducible to SAT.
Since, SAT $\leq$ NP
Then, PSPACE $\subseteq$ NP
So, PSPACE = NP.
Thus, we proved if every NP-hard language is also PSPACE-hard, then PSPACE = NP.

8.16

(a) Set an algorithm as follow:
"On input ϕ:
1. Let each formula φ is shorter than ϕ.
2. Variables of ϕ of each assignment.
   i. If ϕ and φ differs on assignment, then continue go to next φ.
   ii. Else, continue go to next assignment.
   iii. If φ and ϕ are equivalent, reject.
3. If there is no shorter equivalent formula, accept.

In this algorithm, the space can store one formula and one assignment
The total space usage is linear.
So, linear is in the size of ϕ in this algorithm.
Thus, MIN-FORMULA $\in$ PSPACE.

(b) Given: If $\phi \notin$ MIN-FORMULA, then ϕ has a smaller equivalent formula.
$\Rightarrow$ If $\phi \in \overline{\text{MIN-FORMULA}}$, then some smaller formula φ is equivalent to ϕ.
Since, we can not check it within polynomial time.
So, φ can not be an NP.
Check if two formula are equivalent that requires exponential time.

8.25

Let G is a bipartite graph.

Assume G contain a cycle that has odd of nodes $\{n_1, n_2, n_3 \cdots n_{2k}, n_{2k+1}\}$

By defination of bipartite graph. G is a bipartite, If n_1 is in set A. n_2 in set B, then n_3 must in A etc.

All nodes with odd subscript must in A. all the nodes with even subscripe must in B.

So, n_1, and n_{2k+1} are both in A.

That's contradiction, because they are connect.

Thus, G does not contain a cycle with odd of nodes.

Then, we assume a graph does not contain any odd cycles.

Take a node, then label it A. Label all of its neighbors B. Label all of their unlabeled neighbors A, ect until nodes are labeled.

Assume this construction caused two adjacent nodes P, Q that have the same label.

$\Rightarrow$ In even of steps, P and Q were reached

$\Rightarrow$ Exclude start node, form start node to P or Q, the traverse of total number of nodes is even.

So, if we include start node, the total number of nodes is odd.

That's contradition, because a graph does not contain any odd cycles.

Thus, the graph is bipartite.

After, we will show that BIPARTITE $\in NL$.

As we know $NL = coNL$.

$\overline{\text{BIPARTITE}} = \{\langle G \rangle \mid G$ is a graph contain an odd cycle$\}$.

Let M be the NTM that decides $\langle G \rangle \mid G$ $\overline{\text{BIPARTITE}}$ in logarithmic space.

M is constructed as follows:

M = "On input $\langle G \rangle$:

1. Set a counter.
2. Nondeterministically select a start node and successor
3. while counter less or equal than the number of nodes.
4. If counter is odd and successor = start, then accept.
5. Otherwise, successor := Nondeterministically select a successor.

6. If counter is bigger than the number of node, then reject.

So, $\overline{\text{BIPARTITE}} \in NL$.

Thus, BIPARTITE $\in NL$.

8.27

Prove STRONGLY-CONNECTED $\in NL$, we need to construct a NTM N_1 that decides $\overline{\text{STRONGLY-CONNECTED}}$ is logarithmic space.

The constrution of N_1 as follow:

N_1 = "On input $\langle G \rangle$:

1. Select two nodes x and y non-deterministically.
2. Run PATH (x, y).
3. If it reject, then accept.
4. Otherwise, reject.

x, y only takes log space, PATH uses only log space.

So, $\overline{\text{STRONGLY-CONNECTED}} \in NL$

Since, $NL = coNL$.

Thus, STRONGLY-CONNECTED $\in NL$

After, we will show that every language in L is log space reducible to STRONGLY-CONNECTED.

Build a NTM N_2 to do this procedure as follow:

N_2 = "On input $\langle G, s, t \rangle$ G is graph, s, t are vertices in G.

1. Copy all of G onto the output tape.
2. For each node i in G.
3. Set a counter for i.
4. Output on edge from i to s
5. Output on edge from t to i.

For this algorithm, we only need log space to store counter for i.

So, we proved STRONGLY-CONNECTED $\in NL$ and every language in L is log space reducible to STRONGLY-CONNECTED. Thus, STRONGLY-CONNECTED. is in NL-complete.

8.29

First, we show that $A_{NFA} \in NL$.

Let N be an NFA and let w be some valid input word.

Assume that N does not contain ε transitions without loss of generality.

Simulate N on input w.

1. One indicating the current position in w.
2. One pointing to the current state of N.

Each transition of N have more than one possibility.

When we run the end of the w the machine accepts iff the current state of N is final.

Then we show that A_{NFA} is NL-hard.

Set a logspace reducton from reachability in directed graphs as follow:

1. Let G be a directed graph.
2. Let s, t be vertices in G.
3. Let N_G be NFA.
   I. If $u \to v \in G$, $u \to v \in N_G$
   II. The only initial state of N_G is s, the only final state of N_G is t.
3. G has a path from s to t iff N_G accepts empty.
4. N_G can be built in logarithmic space.

So, A_{NFA} is NL-complete.