Assignment 12

Nishan Dhoj Karki

12/07/2022

1 (20 points) Consider three languages A, B, and C. Let's assume the following things are true about these languages:

- **A is PSPACE-complete.**
- **A $\leq_p$ B.**
- **B $\leq_p$ C.**
- **C $\in$ PSPACE.**

Prove that C is also PSPACE-complete.

Proof:

Given the languages A, B, and C.
We have, DEFINITION 8.8
A language B is PSPACE-complete if it satisfies two conditions:
1. B is in PSPACE, and
2. Every A in PSPACE is polynomial time reducible to B.
If B merely satisfies condition 2, we say that it is PSPACE-hard.

Given,
A $\leq_p$ B and B $\leq_p$ C.
By DEFINITION 8.8 as polynomial time reducible,
A $\leq_p$ B $\leq_p$ C.
From the above assumption we can deduce,
A $\leq_p$ C.

Given,
A is PSPACE-complete.
So, BY DEFINITION 8.8, A is PSPACE-hard.
Since, A is PSPACE-hard
By DEFINITION 8.8, C is PSPACE-hard as A $\leq_p$ C.

Given,

C $\in$ PSPACE.
Since, C $\in$ PSPACE and C is PSPACE-hard, we can deduce that C is PSPACE-complete using the DEFINITION 8.8.

2 (20 points) Problem 8.6 from Sipser. 8.6 Show that any PSPACE-hard language is also NP-hard.

Proof idea:

We can show that a PSPACE-hard language is also NPhard if we can show that any language in NP can be efficiently reduced to PSPACE problem.

Proof:

Let us assume that a language A is PSPACE-hard.
Thus by definition of PSPACE-hard, $A_1 \in$ PSPACE, $A_1 \leq_p$ A.
We know, SAT $\in$ NP.
We know, NP $\subseteq$ PSPACE.
Thus, SAT $\in$ PSPACE.
Now, if SAT is in PSPACE it must satisfy, SAT $\leq_p$ A.
If this holds true then A is NP-hard because if any language in NP is efficiently reduced to A then A must be NP-hard. Thus, if A is PSPACE-hard then it must be NP-hard.
Hence, it is proved that PSPACE-hard language is also NP-hard.

3 (30 points) Problem 8.11 from Sipser. Show that if every NP-hard language is also PSPACE-hard, then PSPACE = NP.

Proof:

We have,
NP $\subseteq$ PSPACE.

When PSPACE $\subseteq$ NP and NP $\subseteq$ PSPACE, PSPACE = NP is valid.

To prove PSPACE = NP, PSPACE $\subseteq$ NP should be held.

From (2),
We have,
SAT which is NP-complete.
SAT is NP-hard.
Then, SAT is PSPACE-hard.

So, for any language L $\in$ PSPACE, there exists a polynomial time reduction f from L to SAT such that $x \in L \iff f(x) \in SAT$. We know SAT $\in$ NP, so there is a non-deterministic poly-time Turing Machine M which decides SAT. Let us construct a Turing machine T such that:
T = "On input x:
1. Compute $f(x)$
2. Simulate M on $f(x)$
3. If M accepts, $ACCEPT$, else $REJECT$"
Here, T accepts x if and only if $f(x) \in SAT$ and from definition of f we know that, $x \in L \iff f(x) \in SAT$. So, we can say that T decides L as it accepts x if and only if x $\in L$. Also, since $f(x)$ is polynomially bounded as f is a polynomial time computable function, we can say that T also runs in polynomial time and T is a non-deterministic machine as M is a non-deterministic machine. So, we can say that, $L \in NP$ and thus PSPACE $\subseteq$ NP. Hence, if every NP-hard language is also PSPACE-hard, then PSPACE = NP.

4 (30 points) Problem 8.17 from Sipser. 8.17 Let A be the language of properly nested parentheses. For example, (()) and (()(()))() are in A, but)(is not. Show that A is in L.

Proof:

To check if A is in L, let us describe a turing machine M.

M:"On input of w, where w is a sequence of parenthesis:
-Start at left end of the tape, and keep moving right. Set the initial counter value on work tape to 0.
-if the tape head points at "(", add 1 to the work tape and move one step to the right.
-if the tape head points at ")", if value on work tape < 1, reject. Else, subtract 1 from the work tape and move right.
-If tape head reaches end of the character in w, check if work tape value is 0. If yes, accept. Else, reject.

The above described algorithm requires $O(\log n)$ space as the only space required by the algorithm is to store the counter value to be altered on moving the tape head. Here, n is dependent in the length of w i.e n = $|w|$. Hence, A $\in$ L.