

Homework 8

Due: April 20, 2005

Vinod Vaikuntanathan

Readings: Sections 7.4, 7.5

Problem 1: Let A and B be nontrivial languages over an alphabet Σ (that is, not equal to \emptyset or Σ^*). Explain why each of the following is true.

1. If A is NP-complete, $\bar{A} \in NP$ and $B \in NP$, then \bar{B} must be in NP .
2. If $B \in P$ then $A \cap B \leq_P A$.
3. If $B \in P$ then $A \cup B \leq_P A$.
4. If $A \cup B$ is NP-complete, $A \in NP$ and $B \in P$, then A is NP-complete.

Solution 1:

1. **True.** We claim something stronger: If A is NP-complete and $\bar{A} \in NP$, then $NP = coNP$. The claim follows from the following series of observations:

- A is NP-complete means that for all $L \in NP$, $L \leq_P A$. In particular, $B \leq_P A$.
- If $B \leq_P A$, $\bar{B} \leq_P \bar{A}$. (This statement can be easily proven, similar to the proof for many-one reducibility).
- $\bar{B} \leq L$, for some NP-language L (in particular, $L = \bar{A}$). This means, in turn, that $\bar{B} \in NP$.
- Now, note that the above argument holds for any $B \in NP$. Therefore, we have proven that for all $B \in NP$, $\bar{B} \in NP$ too. That is, $NP = coNP$.

2. **True if $A \neq \Sigma^*$.** To see this, fix an element $a \notin A$. We construct a function f that maps $A \cap B$ into A as follows: On input y ,

- Decide if y is in B . This can be done in polynomial time.
- If so, set $f(y) = y$, else set $f(y) = a$.

Now, it could be the case that there is no element $a \notin A$. In that case, $A = \Sigma^*$, and therefore, such a reduction cannot exist.

3. **True if $A \neq \emptyset$.** To see this, fix an element $a \in A$. We construct a function f that maps $A \cup B$ into A as follows: On input y ,

- Decide if y is in B . This can be done in polynomial time.
- If not, set $f(y) = y$, else set $f(y) = a$.

It could be the case that there is no element $a \in A$. In that case, $A = \emptyset$, and therefore, such a reduction cannot exist.

4. **True if $A \neq \emptyset$.**

- Since $B \in P$, by (3), it follows that $A \cup B \leq_P A$, unless $A = \emptyset$.
- Therefore, A is NP-hard.

- Since $A \in NP$ too, A is NP -complete (unless $A = \phi$).
- Now, what if $A = \phi$? We have that $A \cup B = B$ is NP -complete. But since, $B \in P$, we have that $P = NP$. If $P = NP$, then any $A \in NP$ (except $A = \phi$, Σ^* is trivially NP -complete). Since $A = \phi$, it is not NP -complete.

Problem 2: For each of the following pairs of sets A and B , show that $A \leq_P B$.

1. $A = SAT$, and
 $B = TRIPLE - SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula that has at least three distinct satisfying assignments} \}$.
2. $A = VC$, the Vertex Cover problem, and
 $B = HALF-VC$, defined as $\{ \langle G \rangle \mid G \text{ is an undirected graph with an even number of vertices, of which some half form a vertex cover} \}$.

Solution 2:

1. Let the input formula be ϕ defined over a set of variables $V = \{x_1, \dots, x_n\}$. We transform it into a formula ϕ' over a variable set V' as follows:
 - The new formula is defined over $V' = \{x_1, \dots, x_n\} \cup \{y, z\}$.
 - $\phi' = \phi$.

Now, if the initial formula had at least one satisfying assignment, we can augment the assignment by adding any of the 4 possible values for y and z to give at least 4 satisfying assignments. On the other hand, if ϕ is unsatisfiable, so is ϕ' .

2. We give a polynomial reduction from VC to $HALF-VC$. That is, if $\langle G, k \rangle$ is an instance of a VC problem, we turn G into G' so that G' has a half-vertex-cover iff G has a vertex-cover of size k . The mapping given here will depend on k :
 - If $k = \frac{|G|}{2}$, where $|G|$ is the number of nodes in G , then $G' = G$. Nothing more need be done.
 - If $k < \frac{|G|}{2}$, then define $G' = G$. Since G has a vertex-cover of size $k < \frac{|G|}{2}$, it has a vertex cover of size $\frac{|G|}{2}$, formed by adding some $\frac{|G|}{2} - k$ vertices to the vertex cover of size k . (If G is even, we need to add a new node v' that is not connected to any other node, to get G').
 - If $k > \frac{|G|}{2}$, then let $n = 2k - |G| \geq 1$, where $|G|$ is the number of nodes in G . We expand G to G' so that any k -VC will be expanded to a half VC.

To form G' , add n new nodes v_1, \dots, v_n to G . These new nodes are not connected to any other node. This forms our graph G' .

Suppose G had a vertex-cover C of size at most k . This means, C is a vertex cover for G' too, since no new edges were added in creating G' from G . Therefore, G' has size $|G| + 2k - |G| = 2k$, and has a vertex-cover of size k . For the converse, note that the smallest vertex-cover of G' will not consist of any new vertices that we added to create G' from G . Thus, if G' has a vertex-cover of size k , so will G .

Problem 3: (Sipser 7.39) In the proof of the Cook-Levin theorem, a window is defined to be a 2 by 3 rectangle of cells. Show why the proof would have failed if we had used 2 by 2 windows instead.

Solution 3: (Borrowed from Susan Hohenberger, Spring 04)

Generally speaking, trouble can arise when checking left moves of the TM head. In particular, we can show that there exist two rows of configurations, such that the top row is a legal configuration, and the bottom row is not a configuration that could be produced from the top row by the transition function, but at all 2×2 windows are legal.

Consider the following 2×3 window, where the top row is part of a legal configuration (and obviously the bottom row does *not* legally follow – the regular TM cannot be in two states at once).

0	q_1	0
q_2	0	q_3

However, the 2×2 window scheme cannot catch this problem. It only checks the following two windows:

0	q_1
q_2	0

, which could be a valid transition; for example, the result of $\delta(q_1, 1) \rightarrow (q_2, 1, L)$.

The problem is that this window can't *see* whether the head is looking at a 1 or a 0, so it assumes it is valid.

q_1	0
0	q_3

, which could be a valid transition; for example, the result of $\delta(q_1, 0) \rightarrow (q_3, 0, R)$.

If both of these checks pass, then obviously we have a problem.

Problem 4: (Sipser Problem 7.42) A 2cnf-formula is an AND of clauses, where each clause is an OR of at most two literals. Let $2SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 2cnf-formula}\}$. Show that $2SAT \in P$.

Solution 4: Given a 2cnf formula ϕ on variables x_1, \dots, x_n , construct a directed graph G as follows:

- The vertices of G are the variables in ϕ and their negations. $V = \{x_1, \dots, x_n\} \cup \{\bar{x}_1, \dots, \bar{x}_n\}$.
- For every clause of the form $\alpha \vee \beta$, add a directed edge from $\bar{\alpha}$ to β and one from $\bar{\beta}$ to α .

Note: Intuitively, these edges stand for $\bar{\alpha} \Rightarrow \beta$ and $\bar{\beta} \Rightarrow \alpha$, which are equivalent ways to state the given clause.

Note a special property of the graph: If there is an edge from α to β , there is an edge between $\bar{\beta}$ to $\bar{\alpha}$ (This is by construction of the graph).

Now, suppose there is a directed path from (the vertex corresponding to) x_i to (the vertex corresponding to) \bar{x}_i and a directed path from \bar{x}_i to x_i . Then, we claim that the formula ϕ is unsatisfiable. To see this, note that the existence of a directed path from x_i to \bar{x}_i is equivalent to saying that $x_i \Rightarrow \bar{x}_i$. (This can be proven for induction on the length of the path. Note that the base case is true by construction.) Similarly, a directed path from \bar{x}_i to x_i says that $\bar{x}_i \Rightarrow x_i$. Together, they imply that $x_i \equiv \bar{x}_i$, which is false.

Note: Intuitively, what we end up saying here is that, if this condition occurs, then the formula ϕ has an unsatisfiable clause embedded in it.

Conversely, if there is no such pair of paths (one from x_i to \bar{x}_i and another from \bar{x}_i to x_i), then we can find a satisfying assignment by the following algorithm.

- For each variable x_i , check if there is a path from x_i to \bar{x}_i . If there is such a path, assign $x_i = \text{false}$. For all literals ℓ such that there is a path from \bar{x}_i to ℓ , assign ℓ to true and $\bar{\ell}$ to false.
- For each variable x_i , check if there is a path from \bar{x}_i to x_i . If there is such a path, assign $x_i = \text{true}$. For all literals ℓ such that there is a path from x_i to ℓ , assign ℓ to true and $\bar{\ell}$ to false.
- Propagate all the “true values” down the paths, and the “false values” up the paths.

Note that this algorithm is well-defined (i.e, it never tries to assign the same variable both true and false). Why? For each variable x_i , exactly one of the steps (Step 1 or Step 2) get executed. This is because there does not exist a path from x_i to \bar{x}_i and one from \bar{x}_i to x_i . Moreover, suppose (in Step 1) that there is a path from literal \bar{x}_i to ℓ and from \bar{x}_i to $\bar{\ell}$. This means (by the special property of the graph) that there is a path from \bar{x}_i to $\bar{\ell}\ell$ and therefore, one from \bar{x}_i to x_i . But, we already know that there is a path from x_i to \bar{x}_i . This gives us a contradiction. Thus, no literal has the possibility of being assigned both true and false, and therefore, we get a consistent satisfying truth assignment.

Problem 5: (Sipser 7.37) Show that, if $P=NP$, it is possible to factor positive integers into their prime factors in polynomial time. (Note: NP is a class of *languages* and here, you are being asked for an *algorithm* that produces a factorization for a given integer (as opposed to deciding a language). Thus, simply saying that, “because non-primality is in NP, you are done” isn’t enough.)

Solution 5: Define the language L as follows:

$$L = \{\langle n, k \rangle \mid n \text{ has a prime factor bigger than } k\}$$

L is clearly in NP . Thus, we can determine for a given n and k , whether n has a prime factor larger than k in (deterministic) polynomial time (since, by assumption, $P = NP$).

Our goal is, given an n , to find its prime factorization in polynomial time. Given an n , the following procedure finds *one of the prime factors of n* :

1. (Binary Search) Set the variables $low = 1$, $high = n$
2. Set $current = \lfloor (low + high) / 2 \rfloor$.
3. For $i = 1$ to $\log n - 1$, do:
 4. determine if $\langle n, current \rangle \in L$. (Note: This can be done in polytime by assumption.)
 5. If so, set $low := current$, else set $high := current$.
 6. If $low = high - 1$, output $high$ as one of the prime factors of n . Else, go to Step 2.

We claim that the procedure, as above, finds the largest prime factor of n . The correctness of the procedure is left as an exercise to the reader. (Intuitively, note that if p is the largest prime factor of n , then at any point of time, $low < p$ and $high \geq p$. Also, every step of the procedure cuts down the ratio between low and $high$ by a half).

After executing this procedure on n , we get a prime factor p of n . In fact, this procedure gives us the largest prime factor of n . Thus, we are left with determining the prime factorization of $\frac{n}{p} \leq \frac{n}{2}$. Run the above procedure on $\frac{n}{p}$ next, and continue thus until we get a complete factorization of n .

In one execution of the above algorithm, we reduced finding the prime factorization of n to finding the prime factorization of a number which is at most $n/2$. Thus, we would need to run the above procedure at most $\log n$ times before factoring n completely. How much does it cost to run the above algorithm once? We have to determine membership in L at most $\log n$ times. The complexity is thus, $O(\log n \text{ time}(L))$, where $\text{time}(L)$ is the time required to decide membership in L . The total time complexity is, therefore at most $O(\log^2 n \times \text{time}(L))$, which is polynomial in $\log n$ as long as $\text{time}(L)$ is.