COMS 4236 Homework 6

Mengqi Zong < *mz2326@columbia.edu* >

April 30, 2012

Problem 1

For any input x, let the Boolean expression in disjunctive normal form in the input be d. We first guess a Boolean expression c in conjunctive normal form that has at most B clauses. Let a be an assignment to the Boolean expression. Then $(d-c)(a)$ denote that the value of expression $d-c$ for assignment a. We now could show that the language L defined by this problem is

$$L = \{x : \exists\, c\, \forall a \text{ such that } (d-c)(a) = 0\}$$

Since B is given in unary and c has at most B clauses, we can verify if $(d-c)(a) = 0$ for any a in $poly(x)$ time. Then by the definition of the certificate version of Σ_2, we know that $L \in \Sigma_2$. So this problem is in Σ_2.

Problem 2

1. It's easy to show that DFA Intersection problem is in NPSPACE. We simply guess a string by first guessing its length and then guessing every tape cell of the string. Then verify if this string can be accepted by all n DFAs. Trivially, we can use n work tapes and each tape that initially stores the string x. So, this problem is in NPSAPCE. By Savitch's Theorem, this problem is in PSPACE.

2. We will show that the DFA Intersection problem is PSPACE-hard by reducing LBA problem to this problem.

Given an input x to the LBA, we will construct the DFAs as follows:
Let $n = |x| + 2$ (including endmarkers), we construct n DFAs, $A_1, ..., A_n$. Suppose the LBA has a transition $(q, a, i) \to (p, b, j)$. Then for all DFA except A_i, A_j, they will have a transition $(a, i) \to (b, j)$. As to A_i, it will have a transition $(p, a, i) \to (b, j)$. As to A_j, it will have a transition $(a, i) \to (q, b, j)$. And when LBA accepts, mark all DFAs corresponding states accepting.

As we can see, all DFAs are different, and LBA accepts x if and only if all DFAs accepts x. So, the DFA intersection problem is PSPACE-hard.

To sum up, the DFA intersection problem is PSPACE-hard.

Problem 3

1. It is trivial that #2SAT is in #P. Now we will prove that #2SAT is #P-hard by reducing #Matchings to #2SAT.

For any #Matchings problem, given an undirected graph G, we construct a 2SAT monotone formula as follows:

- For each edge (i,j) in G, we construct a Boolean variable.
- For each pair of edges that share the same node, we create a new term in the 2SAT.

  For example, suppose there are three edges that connect to node 1: (1,2), (1,3), (1,4). And we use Boolean variables v_2, v_3, v_4 to represent each edge. Since there are $\binom{3}{2} = 3$ pairs of edges, we create 3 new terms: $(v_2 \vee v_3)$, $(v_2 \vee v_4)$, $(v_3 \vee v_4)$.
- Combine all terms together with $\wedge$, we get the #2SAT monotone formula ϕ.

Now we will show that the number of satisfying assignments of ϕ equals to the number of matchings in G. For every satisfying assignment of ϕ, for each variable v_i, if $v_i = 1$, then the edge represented by v_i is not in the matching. That is, all the edges whose $v_i = 0$ form a matching of G. We will show that all those edges don't share the same node. If there is one pair of edges share the same node, since all edges in the matching is equivalent to a variable $v_i = 0$, then one term in the #2SAT will not be satisfied,

then the formula ϕ will not be satisfied. This contradicts with the fact that the assignment is a satisfying assignment of ϕ. So, #2SAT is #P-hard.

To sum up, #2SAT is #P-complete.

2. It is trivial that #Node Covers is in #P. We will prove that #Node Covers is #P-hard by reducing #2SAT to #Node Covers.

For any #2SAT problem, given a 2SAT formulas ϕ, we will create a undirected graph G as follows:

- For every variable v_i in ϕ, we create a node n_i in G.
- For every term $(v_i \vee v_j)$, we create an edge (n_i, n_j) in G.

Now we will show that the number of node covers of G equals to the number of satisfying assignment of ϕ. For every set of nodes that cover all edges in G, since each edge in G represents a term in the #2SAT formula ϕ, then covering all nodes means all terms in ϕ are satisfied. So, each node cover of G is equivalent to a satisfying assignment to ϕ. And each nodes in the node cover means the variable represented by this node is 1. So, #Node Covers is #P-hard.

To sum up, #Node Cover is #P-complete.