CISC 462 — Computability and Complexity
Problem Set 3
Solutions

Queen's University, Winter 2017

1. Given a DFA A with n states, we want to determine whether $L(A) = \Sigma^*$. We will use the solution to Question 5 from Assignment 1. First, we construct a DFA A' that recognizes the complement of $L(A)$. In other words, $L(A') = \overline{L(A)}$. To do this, we make all non-accepting states of A into accepting states in A' and make all accepting states in A be non-accepting states in A'. This takes $O(n)$ time. Next, we whether $L(A')$ is empty. This algorithm is described as E_{DFA} and involves marking at most n states. This takes $O(n)$ time. Thus, ALL_{DFA} takes $O(n) + O(n) = O(n)$ time in total and thus, $ALL_{DFA} \in \mathbf{P}$.

2. Let $A \in \mathbf{P}$ and $A \neq \emptyset, \Sigma^*$. Since we know $A \in \mathbf{P}$, we want to show that A is **NP**-hard. Let B be some **NP**-complete problem and we show that $B \leq_P A$. Since A is neither $\emptyset$ or Σ^*, there exist words $x \in A$ and $y \notin A$. This gives us the following reduction:

$$f(w) = \begin{cases} x & \text{if } w \in B, \\ y & \text{if } w \notin B. \end{cases}$$

   Because $\mathbf{P} = \mathbf{NP}$, we have a polynomial time algorithm that decides B. Thus, $B \leq_P A$ and A must be **NP**-complete.

3. First, we show that DOUBLE-SAT is in **NP**. An appropriate certificate is two different satisfying assignments. This has length $O(n)$ where n is the number of variables.

   Next, we show that DOUBLE-SAT is **NP**-hard by reducing from 3SAT. Given a 3CNF formula φ, we want to construct a new formula ψ such that ψ has two satisfying assignments if and only if φ has a satisfying assignment. Our reduction is the following: set $\psi = \varphi \wedge (z \vee \overline{z})$ for some new variable z. This is clearly polynomial in the size of φ. Now, we simply need to show correctness.

   Suppose φ is satisfiable and let $\mathcal{A}$ be a satisfying assignment for φ. Then we have two satisfying assignments for ψ: $\mathcal{A} \cup \{z = 1\}$ and $\mathcal{A} \cup \{z = 0\}$. Now suppose that ψ has two satisfying assignments $\mathcal{A}$ and $\mathcal{B}$. Then this means that $\mathcal{A}$ has an assignment which causes every clause of ψ to evaluate to True. Since $\psi = \varphi \wedge (z \vee \overline{z})$, this means that $\mathcal{A}$ has an assignment that causes every clause of φ to evaluate to True. Thus, φ is satisfiable if and only if ψ has two satisfying assignments.

4. First, we verify that DOMINATING-SET is in **NP**. Given the dominating set as a certificate, we can verify that the set is of size k and that every node is in the set or shares an edge with some vertex in the set in polynomial time.

Now, we show that it is **NP**-complete by giving reduction from VERTEX-COVER. Given a graph $G = (V, E)$ we want to construct a graph $H = (V', E')$ so that H has a dominating set of size k if and only if G has a vertex cover of size k.

Let $V' = V \cup V_E$ be the vertex set of H, which contains all original vertices $v \in V$ and new vertices $v_e \in V_E$ for each edge $e \in E$. Let $E' = E \cup E_E$, which contains all edges $e \in E$ and for each edge $e \in E$ with $e = (u, v)$, we add the edges (u, v_e) and (v, v_e), connecting each new vertex v_e to the endpoints of its corresponding edge. The construction of H can be done in time polynomial in the size of G since $|V'| = |V| + |E|$ and $|E'| = 3|E|$.

Suppose that G has a vertex cover C of size k. Since C is a vertex cover, every edge in G is adjacent to some vertex in C. Thus, in the graph H, we observe that each vertex $v \in C$ shares an edge with some other vertex in V. In addition, for each $v \in C$ and each edge $e \in E$ that has v as an endpoint, v is also adjacent to the vertex v_e. Thus, every vertex in V' is adjacent to some vertex in C and C is a dominating set in H of size k.

Now, suppose that H contains a dominating set D of size k. We observe that any dominating set that contains a vertex $v_{(u,v)} \in V_E$ can be modified so that either u or v is in the dominating set instead of $v_{(u,v)}$. To see this, we have $v_{(u,v)}$ dominating u and v, but, without loss of generality, suppose we change the dominating set to contain v instead. Then v dominates $v_{(u,v)}$ and u, preserving the dominating property of the dominating set. Thus, we can safely assume that D contains only vertices from V. Now, we claim that D is a vertex cover for G. Consider any edge $e \in E$. We note that because D is a dominating set, there exists a vertex $v \in D$ that dominates v_e in H. This means that v is an endpoint of e and thus, covers e in G. Thus D is a vertex cover for G of size k.

5. To see that L is in **DP**, we define the languages

$$L_1 = \{\langle G_1, k_1, G_2, k_2\rangle \mid G_1 \text{ has a } k_1\text{-clique}\},$$
$$L_2 = \{\langle G_1, k_1, G_2, k_2\rangle \mid G_2 \text{ doesn't have a } k_2 \text{ clique}\}.$$

Then we can see that L_1 and $\overline{L_2}$ are essentially CLIQUE and are in **NP**. We then have $L = L_1 \cap L_2 \in \mathbf{DP}$.

Now we show that every language in **DP** is polynomial time reducible to L. Consider an arbitrary language $K \in \mathbf{DP}$. By definition, we can write $K = K_1 \cap K_2$ such that $K_1 \in \mathbf{NP}$ and $\overline{K_2} \in \mathbf{NP}$. To construct a reduction, we observe that since CLIQUE is **NP**-complete and K_1 and $\overline{K_2}$ are in **NP**, they are both polynomial time reducible to CLIQUE. Let f_1 be the reduction for $K_1 \leq_P$ CLIQUE and let f_2 be the reduction for $\overline{K_2} \leq_P$ CLIQUE. In other words, we have

$$w \in K_1 \iff f_1(w) = \langle G_1, k_1\rangle \in \text{CLIQUE},$$
$$w \in \overline{K_2} \iff f_2(w) = \langle G_2, k_2\rangle \notin \text{CLIQUE}.$$

Then we define the reduction $f(w) = \langle f_1(w), f_2(w) \rangle$. This gives us $K \leq_P L$ as desired and thus, L is **DP**-complete.

6. Let φ be a 2CNF formula

$$\varphi = (a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge \cdots \wedge (a_n \vee b_n).$$

We observe that for each clause (a_i, b_i) if the clause is satisfied, we have the implications $\overline{a_i} \to b_i$ and $\overline{b_i} \to a_i$. The idea is that if one of the literals in the clause evaluates to False, the other must evaluate to True in order for the clause to be satisfied. We will use this observation to build a directed graph $G = (V, E)$ as follows:

- for each variable x in φ, we add the nodes x and $\overline{x}$,
- for each clause $(a_i \vee b_i)$, add the edges $(\overline{a_i}, b_i)$ and $(\overline{b_i}, a_i)$.

We claim that there exists a satisfying assignment for φ if and only if G has no directed cycle containing both x and $\overline{x}$ for some $x \in V$. In other words, there is no path from x to $\overline{x}$ and from $\overline{x}$ to x.

First, suppose that there exists a satisfying assignment for φ and suppose for contradiction that there is a path from x to $\overline{x}$ and a path from $\overline{x}$ to x. Recall that an edge in the graph is some implication $a \to b$. If we assign x to True, then on the path from x to $\overline{x}$, there is some implication $a \to b$, where a is True and b is False, which means that a clause is unsatisfied if x is assigned to be True. However, if x is False, then we run into the same problem since there is a path from $\overline{x}$ to x. Thus, both of these paths cannot exist if φ is satisfiable.

Next, suppose that there is no directed cycle containing both x and $\overline{x}$ for some vertex x. We can find a satisfying assignment by the following algorithm:

1. For each unassigned variable x, check if there is a path from x to $\overline{x}$.
2. If not, then set x to True. If there is, set x to False.

We get a satisfying assignment from this. Suppose for contradiction that we didn't and there is some unsatisfied clause $(x \vee y)$. Then that means both x and y were false and so there was a path from x to $\overline{x}$ and a path from y to $\overline{y}$. But the clause $(x \vee y)$ would have added edges $\overline{x} \to y$ and $\overline{y} \to x$ to the graph. But this means that we now also have a path from $\overline{x}$ to x, which is a contradiction.

To see that this problem is in **P**, we first note that from a formula φ with n variables, we create a graph with $2n$ vertices. Recall that we can check whether there is a directed path in time $O(m)$ for a graph with m vertices. We check two paths for each variable in φ: one from x to $\overline{x}$ and another from $\overline{x}$ to x. This gives us $2n$ paths to check. Thus, we have an $O(n^2)$ time algorithm for solving 2SAT.