CS5311 Homework #4 Solution (Spring 2002)

Due: Tuesday, April 16, 2002 **before class**

100 points

1. For each of the following sets, use Rice's Theorem and its extensions to determine if it is recursive, r.e. but not recursive, or not r.e. Prove your claim; otherwise, you will receive **no** credit. Also make sure you cite the known facts properly.

   (a) [**8 points**] $A_1 = \{ x \mid \text{Range } \Phi_x \text{ is finite} \}$
   Answer: Let Γ be the collection of programs $\Gamma = \{ \Phi_x \mid \text{Range}(\Phi_x) \text{ is finite} \}$. Then, A_1 is the index of Γ. A_1 is not r.e. Let θ be defined as follows:

   $$\theta(x) = \begin{cases} 0 & x = 0 \\ \uparrow & \text{otherwise} \end{cases}$$

   θ is partially computable and has a range of exactly one element $\{0\}$. Therefore, $\{ x \mid \Phi_x = \theta \} \subseteq A_1$ holds. The identity function, $\hat{\theta}(x) = x$, is an extension of θ and $\{ x \mid \Phi_x = \hat{\theta} \} \subseteq \bar{A}_1$ holds because the range of $\hat{\theta}$ is $\mathcal{N}$. Consequently, A_1 is not r.e. by the first extension of Rice's Theorem. ∎

   (b) [**8 points**] $A_2 = \{ x \mid W_x = \{ x \} \}$
   Answer: $W_x = \{x\}$ means x is only defined on itself. Let $\Gamma = \{ \Phi_x \mid \Phi_x(x)\downarrow \ \& \ |\text{Domain}(\Phi_x)| = 1 \}$. Then, A_2 is the index set of Γ. We should first prove that A_2 is non-empty by finding a function in W_x; otherwise, $A_2 = \emptyset$ is recursive.
   Consider the following function:

   $$g(y, x) = \begin{cases} 1 & \text{if } x = y \text{ and } y \in K \\ \uparrow & \text{otherwise} \end{cases}$$

   Function $g(y, x)$ is partially computable, and, by the Recursion Theorem, there is a e such that $\Phi(x, e) = g(e, x)$ for all x. By the following reasoning, we know that e is only defined on itself and hence $W_e = \{e\}$ and $e \in A_2$.

   $$\begin{aligned} \forall_x[\Phi(x, e)\downarrow] &\Leftrightarrow \forall_x[g(e, x)\downarrow] \\ &\Leftrightarrow e = x \ \& \ e \in K \end{aligned}$$

   Now take this e and define its extension $\hat{e}$ to be $\hat{e}(x) = 1$. Please verify that $\hat{e}$ is an extension of e. Since $\hat{e}$ is total, $\{ x \mid \Phi_x = \hat{e} \} \subseteq \bar{A}_2$. Therefore, A_2 is not r.e. by the first extension of Rice's Theorem. ∎

   (c) [**8 points**] $A_3 = \{ (x, y) \mid \text{for every } a \in \text{Domain}(\Phi_x) \cap \text{Domain}(\Phi_y),\ \Phi_x(a) < \Phi_y(a) \}$
   Answer: It is very easy to prove A_3 is not recursive with the third extension of Rice's Theorem. To reach a stronger result, we need to do more. Define set A_3^* as follows:

   $$(x, y) \in A_3 \Leftrightarrow \langle x, y \rangle \in A_3^*$$

   It is not difficult to show that A_3 is r.e. if and only of A_3^* is r.e. I will leave the proof to you. Next, we shall prove A_3^* is not r.e. Define functions f and g as follows:

   $$f(x) = \begin{cases} 0 & x = 0 \\ \uparrow & \text{otherwise} \end{cases} \quad \text{and} \quad g(x) = \begin{cases} 1 & x = 0 \\ \uparrow & \text{otherwise} \end{cases}$$

Function $\theta = \langle f, g\rangle$ is a member of A_3^* because $\Phi_f(0) < \Phi_g(0)$ holds for the only element $x = 0$ in the common domain of f and g. Let us extend θ by extending f and g as follows:

$$\hat{f}(x) = \begin{cases} 0 & x = 0 \\ 1 & \text{otherwise} \end{cases} \quad \text{and} \quad \hat{g}(x) = \begin{cases} 1 & x = 0 \\ 0 & \text{otherwise} \end{cases}$$

Obviously, $\hat{f}$ and $\hat{g}$ are extensions of f and g, respectively. Therefore, $\hat{\theta} = \langle \hat{f}, \hat{g}\rangle$ is an extension of $\theta = \langle f, g\rangle$. This is because $\theta(0) = \langle f(0), g(0)\rangle = \langle 0, 1\rangle = \hat{\theta}(0)$. However, since the extension does not satisfy the requirement of $\Phi_{\hat{f}}(x) < \Phi_{\hat{g}}(x)$ for all elements in $\mathcal{N}$ but $x = 0$, $\hat{\theta}$ is not a member of A_3^*. Consequently, A_3^* is not r.e. by the first extension of Rice' Theorem, and A_3 is not r.e. ■

(d) [**8 points**] $A_4 = \{\ x \mid \text{Range}(\Phi_x) = \mathcal{N}\ \}$

Answer: Let Γ be the collection of all programs Φ_x whose range cover the whole natural integers. Then, A_4 is the index set of Γ.

Let θ be an index whose program has $\mathcal{N}$ as its range. Then, any restriction of θ to a finite subset can only produce a finite range, because a function can only map a finite set to another finite set. As a result, any finite restriction cannot be in A_4, and, A_4 is not r.e. by Rice-Shapiro Theorem. ■

(e) [**8 points**] $A_5 = \{\ x \mid a \notin \text{Range}(\Phi_x)\ \}$ for a given fixed a.

Answer: Let Γ be the collection of programs whose ranges do not contain the given fixed value a. Then, A_5 is the index set of Γ. We want to prove that A_5 is not r.e.

Consider function θ defined as follows:

$$\theta(x) = \begin{cases} a+1 & x = 0 \\ \uparrow & \text{otherwise} \end{cases}$$

Then, θ is partially computable and does not have a in its range. Now, let us extend θ to $\hat{\theta}$ as follows:

$$\hat{\theta}(x) = \begin{cases} a+1 & x = 0 \\ a & \text{otherwise} \end{cases}$$

Thus, $\hat{\theta}$ does contain a in its range and is not part of the index set A_5. As a result, A_5 is not r.e. by the first extension of Rice's Theorem. ■

2. [**10 points**] Consider the following proof of **P**$\neq$**NP**.

   (a) We shall prove by contradiction.

   (b) Assume that **P**=**NP**.

   (c) Then, SAT is in **P**, and there is a k such that SAT can be solved in $O(|x|^k)$ time, where x is an input to SAT.

   (d) Since SAT is **NP**-complete, every problem in **NP** can be reduced to SAT in polynomial time and be solved in $O(|x|^k)$. Thus, we conclude that **NP** is a subset of **P**.

   (e) However, since we know that there are problems in **P** that requires a time complexity of at least $O(|x|^{k+1})$, we have a contradiction. Therefore, the assumption is incorrect, and **P**$\neq$**NP**.

What is wrong with this proof? Clearly identify the problem(s) and elaborate your findings.

Answer: The claim in Step (d) is incorrect. Given a problem Q, it takes $O(|x|^m)$ steps to reduce Q to SAT and another $O(|f(x)|^k)$ steps to solve the instance $f(x)$, where f is the reduction function.

Therefore, the combined number of steps is at least $O(|x|^{m+k})$ which can be arbitrary depending on the problem Q. Consequently, the number of steps, as used in Step (e), is not bounded by $O(|x|^{k+1})$. ∎

3. [**12 points**] Given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, G_1 is *isomorphic* to G_2, denoted as $G_1 \cong G_2$, if there is a one-to-one function $f : V_1 \to V_2$ such that for all $u, v \in V_1$, there is an edge between u and v if and only if there is an edge between $f(u)$ and $f(v)$. Consider the following problem:

Problem: Given two undirected graphs $G = (V, E)$ and $H = (W, F)$, is there a subgraph of $H = (W, F)$ that is isomorphic to G?

Show that this problem is **NP**-complete.

Answer: This is actually a very simple problem. The k-clique problem asks for a complete subgraph of size k. Thus, given a graph $G = (V, E)$ and k, the k-clique problem is equivalent to determining if a complete graph of k vertices is isomorphic to a subgraph of G. Hence, we can reduce the k-clique problem to this subgraph isomorphism problem.

First of all, it is not difficult to see that the subgraph isomorphism problem is **NP** because we can guess $|W|$ vertices from V and verify if the subgraph is isomorphic to H.

Given a graph $G = (V, E)$ and k, construct $H = (W, F)$ as follows:

- If $k > |V|$, let $W = V \cup \{x_{|V|+1}\}$ and $F = E$, where $x_{|V|+1}$ is vertex not in V.
  If $k > |V|$, no k-clique can be found in G and this construction makes finding a subgraph that is isomorphic to $H = (W, F)$ impossible because H has more vertices than G has.
- Otherwise, let $W = \{x_1, x_2, \ldots, x_k\}$ and $F = \{\overline{x_i x_j} \mid 1 \le i < k \;\&\; i < j \le k\}$.
  In this case W is a complete graph of k vertices.

We shall show that if G has a k-clique if and only if G has a subgraph that is isomorphic to W. If G has a k-clique, this k-clique is a complete graph of k vertices and is isomorphic to $H = (W, F)$. Conversely, if G has a subgraph that is isomorphic to H which is a complete graph of k vertices, G certainly has a k-clique. Thus, the k-clique problem reduces to the subgraph isomorphism problem and the latter is **NP**-complete. ∎

4. [**10 points**] Is 2SAT in CNF in **P** or **NP**-complete? Make your claim first and prove your claim. If you believe 2SAT is **NP**-complete, show the reduction and prove its correctness. If you believe 2SAT is in **P**, design an algorithm to solve it, prove its correctness and analyze its time complexity.

Answer: 2SAT is in **P**. A clause of form $\bar{p} \vee q$ plays an extremely important role in 2SAT, because assigning *true* to p forces q to be true. In fact, every clause can be converted to this form with de Morgan's law. Suppose p and q are two variables. Then, $p \vee q = \overline{(\bar{p})} \vee q$ means assigning *true* to $\bar{p}$ forces q to be *true*, and $p \vee \bar{q} = \overline{(\bar{p})} \vee \bar{q}$ means assigning *true* to $\bar{p}$ forces $\bar{q}$ to be *true*. Clause $\bar{p} \vee \bar{q}$ is already in this form because it can be interpreted as assigning *true* to p forces $\bar{q}$ to be *true*. The form of $\bar{p} \vee q$ is equivalent to $p \Rightarrow q$ which has an interpretation identical to the above mentioned one (*i.e.*, if p is *true* than q is *true*). With this knowledge in hand, to satisfy the given 2SAT CNF Boolean expression ϕ, we start with p_1 and a sequence of clauses $p_1 \Rightarrow p_2$, $p_2 \Rightarrow p_3$, …, $p_{k-1} \Rightarrow p_k$. If such a sequence can be found, assigning *true* to p_1, forces all p_2, p_3, …, p_{k-1} and p_k to be *true*. Then, we can find another unassigned p_i and repeat this process. If no p_i receives conflict assignments, the given Boolean expression ϕ in 2SAT CNF is satisfiable. This algorithm is definitely in **P**. However, we can do more as follows.

We can use a directed graph to precisely represent the idea of truth value assignment. The set of vertices V contains all variables and their negations. The arcs are defined as follows:

- **The Trivial Case:**
  If $\bar{p} \vee q$ is a clause, since assigning *true* to p forces q to be *true*, we draw an arc $p \to q$.
- **The Commutative Case:**
  Since $\bar{p} \vee q = q \vee \bar{p} = \overline{(\bar{q})} \vee \bar{p}$, the fact that assigning *true* to p forces q to be *true* can also be represented by $\overline{(\bar{q})} \vee \bar{p}$. Therefore, we add an arc from $\bar{q} \to \bar{p}$, which means assigning *true* to $\bar{q}$ forces $\bar{p}$ to be *true*. This is the way of doing "prove by contradiction."

In summary, if $\bar{p} \vee q$ is a clause, two arcs are added to the graph: $p \to q$ and $\bar{q} \to \bar{p}$. As pointed out in the beginning of this proof, since $p \vee q = \overline{(\bar{p})} \vee q$, $p \vee q$ produces two arcs $\bar{p} \to q$ and $\bar{q} \to p$. Similarly, clause $\bar{p} \vee \bar{q}$ produces arcs $p \to \bar{q}$ and $q \to \bar{p}$. Note that two different clauses produces two different sets of arcs. Prove it yourself. Hence, if we have m clauses in n variables, the constructed graph has $2n$ vertices and $2m$ arcs. Note also that because we add two arcs $p \to q$ and $\bar{q} \to \bar{p}$ for each clause $\bar{p} \vee q$, if there is a path from p to q, there is a path from $\bar{q}$ to $\bar{p}$.

The following figure shows the corresponding graph of $(p \vee r) \wedge (\bar{p} \vee q) \wedge (q \vee r) \wedge (q \vee \bar{r})$. Clause $p \vee r$ produces $\bar{p} \to r$ and $\bar{r} \to p$; clause $\bar{p} \vee q$ produces $p \to q$ and $\bar{q} \to \bar{p}$; clause $q \vee r$ produces $\bar{q} \to r$ and $\bar{r} \to q$; and clause $q \vee \bar{r}$ produces $\bar{q} \vee \bar{r}$ and $r \to q$.

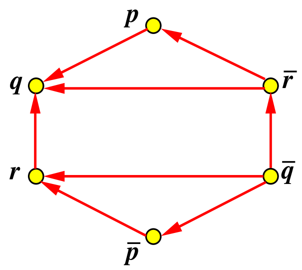

Figure 1: $(p \vee r) \wedge (\bar{p} \vee q) \wedge (q \vee r) \wedge (q \vee \bar{r})$

Consider the graph constructed from a 2SAT CNF Boolean expression ϕ. If ϕ is satisfiable, we will have a path like $p_1 \to p_2 \to p_3 \to \cdots \to p_{k-1} \to p_k$ with which assigning *true* to p_1 forces p_2 to be *true*, which, in turn, forces p_3 to be *true*, and so on. Eventually, p_k is assigned with *true*. However, if there are x and $\bar{x}$ on this path, the truth value assignment cannot be successful. For example, if we have $p_1 \to \cdots \to x \to \cdots \to \bar{x} \to \cdots \to p_k$, x and $\bar{x}$ will receive *true* which is a contradiction. Hence, we see that if there is a path from x to $\bar{x}$ (or from $\bar{x}$ to x), ϕ is not satisfiable. In fact, this is a necessary and sufficient condition.

The given 2SAT CNF Boolean expression ϕ is not satisfiable if and only if there exists a variable x such that there is a path from x to $\bar{x}$ (and a path from $\bar{x}$ to x due to symmetry)

($\Rightarrow$) Suppose ϕ is satisfiable and there is a path from x to $\bar{x}$. On this path, there must be an arc $p \to q$ such that p is *true* and q is *false*. This implies that the corresponding clause $\bar{p} \vee q$ has a truth value of $\overline{\mathit{true}} \vee \mathit{false} = \mathit{false}$. Hence, ϕ is not satisfiable, which contradicts the assumption. Consequently, if ϕ is satisfiable, there is no path from any x to $\bar{x}$.

($\Leftarrow$) Suppose there is no variable x with a path from x to $\bar{x}$. We want to prove ϕ is satisfiable. Consider the following algorithm:

while there are vertices without assigned truth values **do**
  begin
    pick a node p such that there is no path from p to $\bar{p}$;
    apply a depth-first search from p;
    if q is reachable from p **then**
      begin
        assign *true* to q;
        assign *false* to $\bar{q}$;
      end
  end

There are two possibilities that a vertex may receive contradictory values. We shall show both possibilities cannot happen:

- **Two vertices are assigned contradictory values in the *same* DFS iteration.**
  Suppose there is a q that receives two values (*i.e.*, both *true* and *false*). There is a path from p to q and a path from p to $\bar{q}$. Since there is a path from p to q, there is a path from $\bar{q}$ to $\bar{p}$ due to symmetry. Since there is a path from p to $\bar{q}$, there is a path from q to $\bar{p}$. Both conditions yield a path from p to $\bar{p}$, which contradicts our assumption. Thus, each variable receives exactly one truth value in the same DFS iteration.
- **Two vertices are assigned contradictory values in *different* DFS iterations.**
  Suppose there is a path from p to q and q is assigned with *false* in a previous iteration. Since there is a path from p to q (found in this DFS), there is a path from $\bar{q}$ to $\bar{p}$. If q was assigned a *false* previously, $\bar{q}$ was assigned *true* in that previous iteration. As a result, after $\bar{q}$ is assigned with *true*, $\bar{p}$ is assigned with *true* when the path from $\bar{q}$ reaches $\bar{p}$ in a previous DSF. Therefore, before this iteration starts, p has already received a truth value. This is a contradiction because p must have no value in order to start this iteration.

In this way, every vertex of the graph receives a unique truth value such that p and $\bar{p}$ receive different values. Each assignment makes a clause satisfiable, and ϕ is satisfied. The algorithm is clearly polynomial because DFS is applied at most $2n$ times and each application of DFS has a time complexity of $O(m + n)$, where m and n are the number of clauses and the number of variable names. As a result, the time complexity is at most $O(n(m + n))$ which is obviously polynomial. ∎

5. [**12 points**] Consider the following problem:

> Given a family of sets S_1, S_2, …, S_n, is there a subfamily of k sets S_{i_1}, S_{i_2}, …, S_{i_k} such that the following holds?
>
> $$\cup_{j=1}^{k} S_{i_j} = \cup_{j=1}^{n} S_j$$
>
> Note that all sets are finite sets whose members may not be integers.

Show that this problem is **NP**-complete by reducing the vertex cover problem to it.

Answer: This is the so-called *set cover* problem. It is in **NP** because we can guess k distinct values i_1, i_2, …, i_k from the set $\{1, 2, \ldots, n\}$ and verify if $\cup_{j=1}^{k} S_{i_j} = \cup_{j=1}^{n} S_j$ holds.

Next, we shall reduce the vertex cover problem to the set cover problem. More precisely, given a graph $G = (V, E)$, we want to construct sets S_1, S_2, …, $S_{|V|}$ such that G has a vertex cover of size k if and only if we can find sets S_{i_1}, S_{i_2}, …, S_{i_k} such that $\cup_{j=1}^{k} S_{i_j} = \cup_{j=1}^{n} S_j$ holds.

The following observation is important to the reduction. Suppose $S \subseteq V$ is a vertex cover of G. Since each edge is incident to at least one vertex in S, the union of all edges that are incident to a vertex in S is the set of all edges. Therefore, given a graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$, let S_{v_i} be the set of all edges incident to vertex v_i. This construction requires no more than $O(|V| + |E|)$ time. Now, if $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ is a vertex cover of size k, we have $\cup_{j=1}^{k} S_{i_j} = \cup_{j=1}^{n} S_j = E$. On the other hand, if we have k sets S_{i_1}, S_{i_2}, …, S_{i_k} such that $\cup_{j=1}^{k} S_{i_j} = \cup_{j=1}^{n} S_j = E$ holds, an edge in E belongs to at least one S_{i_j}. Thus, this edge is incident to vertex v_{i_j}, and vertices v_{i_1}, v_{i_2}, …, v_{i_k} form a vertex cover. Since the vertex cover problem is **NP**-complete and the set cover problem is in **NP**, the set cover problem is also **NP**-complete. ∎

6. A set $A \subseteq V$ of a graph $G = (V, E)$ is an *independent set* of size k if and only if $|A| = k$ and for any two vertices u and v in A we have $\overline{uv} \notin E$. The *independent set of size k problem*, IS_k, is

Given a graph $G = (V, E)$ and a positive integer k, determine whether G has an independent set of size k.

The *n-queens problem* nQ is the following:

Place n queens in an $n \times n$ chessboard without them attacking each other, where $n \geq 4$.

Note that a queen on row s and column t can attack all positions on row s, column t, and the two diagonals containing position (s, t).

Do the following problems:

(a) [**10 points**] Show that $n\mathrm{Q} \leq_{\mathrm{p}} \mathrm{IS}_k$.

<u>Answer:</u> Given an $n \times n$ chessboard, we need to construct a graph $G = (V, E)$ such that if this $n \times n$ board has an n-queens solution if and only if graph G has an independent set of size n. The vertex set V of G contains all positions of the chessboard. More precisely, we have

$$V = \{\ v_{pq} \mid v_{pq} \text{ is the position on row } p \text{ and column } q\ \}$$

For each position v_{pq}, we connect it with all positions that v_{pq} can attack. As a result, once v_{pq} is selected, none of the positions v_{pq} can attack will be in the same independent set that contains v_{pq}. Therefore, the edge set E is

$$E = \{\overline{v_{pg}v_{st}} \mid (p = s)\ \vee\ (q = t)\ \vee\ (|p - s| = |q - t| \neq 0) \text{ and } v_{pg} \neq v_{st}\ \}$$

In the definition of E, v_{pq} and v_{st} has an edge connected if $p = s$ (on the same row), or $q = t$ (on the same column), or $|p - s| = |q - t| \neq 0$ (on one of the two diagonals). Now, it is easy to see that the nQ problem has a solution if and only if the graph $G = (V, E)$ has an independent set of size n.

The construction of V takes $O(n^2)$ time because we need to scan all $n \times n$ positions. For each position v_{pq}, there are at most $4n$ edges constructed (*i.e.*, $n - 1$ edges for row p, $n - 1$ for column q, and $2(n - 1)$ for the diagonals), and E has at most $O(n^2(4n)) = O(n^3)$ edges. As a result, the construction (*i.e.*, reducing nQ to ISn) is $O(n^2 + n^3) = O(n^3)$. ∎

(b) [**6 points**] B.-J. Falkowski and L. Schmitz in 1986[1] and M. Reichling in 1987[2] published a set of formalæ that provides *one* solution to the nQ problem. Does this fact imply that the independent

[1]B.-J. Falkowski and L. Schmitz, A Note on the Queens' Problem, *Information Processing Letters*, Vol. 23 (1986), pp. 39–46.

[2]M. Reichling, A Simplified Solution of the N Queens' Problem, *Information Processing Letters*, Vol. 25 (1987), pp. 253–255.

set problem is in **P** rather than **NP**-complete? Elaborate your findings. Providing only an answer receives **no** credit.

Answer: The answer is "no." We only know $n\mathrm{Q} \leq_{\mathrm{p}} \mathrm{IS}_n$ and $n\mathrm{Q}$ is polynomial. This does not imply IS_n is polynomial. Moreover, $n\mathrm{Q}$ reduces to IS_n, which is a special case of IS_k when $k = n$. Both cases do not imply IS_k is in **P**. Note that $A \leq_{\mathrm{p}} B$ implies A in **P** if B is in **P** rather than the other way around. As a result, we cannot deduce IS_n is in **P** if $n\mathrm{Q}$ is in **P** even though we know $n\mathrm{Q} \leq_{\mathrm{p}} \mathrm{IS}_n$. ∎