Homework 8

Yutong Huang (yxh589)

Problem 1

Assume languages $L_1, L_2 \in NP \iff \exists$ deterministic TMs, $V_1(x,y)$ and $V_2(x,y)$, and polynomials p_1, p_2, q_1, q_2 such that $\forall x, y$, V_1 runs in $p_1(|x|)$ time and V_2 runs in $p_2(|x|)$ time $\wedge$
$\forall x \in L_1, \exists y$ such that $|y| = q_1(|x|), V_1(x,y) accepts$ and $\forall x \in L_2, \exists y$ such that $|y| = q_2(|x|), V_2(x,y) accepts \wedge$
$\forall x \notin L_1, \forall y \in \{y \mid |y| = p_1(|x|)\}, V_1(x,y)\ rejects$ and $\forall x \notin L_2, \forall y \in \{y \mid |y| = p_2(|x|)\}, V_2(x,y)\ rejects.$

union $- L_1 \cup L_2 \in NP$

Proof. Construct a verifier TM V_3:

```
function v3(x,y) {
    run v1(x,y);
    if v1 accepts
        accept;
    run v2(x,y)
    if v2 accepts
        accept;
    reject;
}
```

We know that by definition, V_1 and V_2 runs in polynomial times, therefore V_3 runs in polynomial time as well.

Case 1: $x \in L_1 \implies \exists y \in \{y||y| = q_1(|x|)\} such\ that\ V_1\ accepts \implies \exists y \in \{y||y| = q_1(|x|)\} such\ that\ V_3\ accepts$
Case 2: $x \in L_1$
Case 2.1: $x \in L_2 \implies \exists y \in \{y||y| = q_2(|x|)\} such\ that\ V_2\ accepts \implies \exists y \in \{y||y| = q_2(|x|)\} such\ that\ V_3\ accepts$
Case 2.2: $x \notin L_2 \implies$ both V_1 and V_2 reject $\implies$ V_3 rejects.

Therefore V_3 verifies if $x \in L$ with advice string y in polynomial time $\iff L_1 \cup L_2 \in NP$. $\square$

concatenation $- L_1 \circ L_2 \in NP$

Proof. Construct a verifier TM V_4:

```
function v4(x, y) {
    if (y is in the form "k#a#b"){ //k is a number, a and b are strings, # is a new character
        run v1(x[0:k-1], a); //slicing operator represents substrings
        run v2(x[k:], b);
        if (v1 accepts) and (v2 accept){
            accept;
        }
        reject;
    }
    reject;
}
```

Similarly, because both V_1 and V_2 run in polynomial times, V_4 also runs in polynomial time.

Also by definition, we know that if a verifier accepts, then the advice string length is polynomial of the input string.

Case 1: $x \in L_1 \circ L_2 \implies a$, and b are polynomials of $x_1 x_2 \dots x_k$ and $x_{k+1} \dots x_n \implies$ "k#a#b" is polynomial of x. Also V_1, V_2 accpet in this case $\implies V_4$ accepts. Therefore $\forall x \in L_1 \circ L_2, \exists y \in y||y| = p(|x|) \wedge V_4$ accepts.
Case 2: $x \notin L_1 \circ L_2 \implies$ at least one of V_1, V_2 rejects $\implies$ for rejecting machine V_i: $\forall x \notin L_i, \forall y \in y||y| = p(|x|) \wedge V_i$ rejects. $\implies \forall x \notin L_1 \circ L_2, \forall y \in y||y| = p(|x|) \wedge V_4$ rejects.

Therefore, $L_1 \circ L_2 \in NP$. □

Problem 2

Because the Clique Problem is NP-Complete, we can try to reduce the Clique Problem to SUBGRAPH ISOMORPHISM to show that it is in NP.

Claim: $CLIQUE \leq_P SUBGRAPH\ ISOMORPHISM$

Proof. Suppose there is an algorithm `sgi(g: Graph, h: Graph)` that decides if `h` is isomorphic to a subgraph of `g`. Consider the following algorithm that decides if a graph has a clique with a given number of vertices:

```
function clique(g: Graph, k: int){
    generate a complete graph h with k vertices;
    return sgi(g, h);
}
```

Case 1: If `g` has a clique of `k` vertices, then `g` has a subgraph that's isomorphic to `h`, meaning that `sgi(g, h)` returns true, and thus `clique(g, k)` is true;
Case 2: If `g` does not have a clique of `k` verices, then `g` does not have a subgraph that's isomorphic to `h`, meaning that `sgi(g,h)` returns false, and thus `clique(g,k)` returns false;

Therefore, the algorithm is correct.

Assume the input graph has `n` vertices, the input size could be as large as $n^2 + n + 1$. The generation of the complete graph `h` can be performed in $O(k^2)$, which is smaller than $O(n^2)$.

Therefore, this reduction runs in polynomial times $\implies CLIQUE \leq_P SUBGRAPH\ ISOMORPHISM$.

Since $CLIQUE$ is NP-complete, then $SUBGRAPH\ ISOMORPHISM \in NP$.

□

Problem 3

To show that $HAMILTON\ CYCLE \leq_P GRAPH\ ISOMORPHISM$, construct an algorithm that transform the input to `ham_cycle(g: Graph)` to invoke `graph_iso(g: Graph, h: Graph)` and produce the correct result.

Proof. Suppose algorithm `graph_iso(g: Graph, h: Graph)` correctly decides if `g` is isomorphic to `h`. Construct the following algorithm:

```
function ham_cycle(g: Graph){
    k <- count the number of vertices in g;
    h <- create a Hamilton cycle using k vertices;
    return graph_iso(g, h);
}
```

Case 1: g is not a Hamilton cycle, then g is not isomorphic to a Hamilton cycle with `k` vertices, therefore `graph_iso(g,h)` returns false, and thus `ham_cycle(g)` returns false;
Case 2: g is a Hamilton cycle, then g is isomorphic to a Hamilton cycle with `k` vertices, therefore `graph_iso(g,h)` returns true, and thus `ham_cycle(g)` returns true.

Therefore this algorithm is correct.

Assume `g` has n vertices, the number of vertices in `g` could be counted in $O(n)$, the new Hamilton cycle could be constructed in $O(n)$.
Therefore this reduction runs in polynomial time $\implies$ $HAMILTON\ CYCLE \leq_P GRAPH\ ISOMORPHISM$ $\square$