## Homework 12 Solutions

1. (a) Show that NP is closed under union.

## Answer:

Let  $L_1$  and  $L_2$  be languages in NP. Also, for i = 1, 2, let  $V_i(x, c)$  be an algorithm that, for a string x and a possible certificate c, verifies whether c is actually a certificate for  $x \in L_i$ . Thus,  $V_i(x, c) = 1$  if certificate c verifies  $x \in L_i$ , and  $V_i(x, c) = 0$  otherwise. Since both  $L_1$  and  $L_2$  are both in NP, we know that  $V_i(x, c)$  terminates in polynomial time  $O(|x|^d)$  for some constant d.

To show that  $L_3 = L_1 \cup L_2$  is also in NP, we will construct a polynomial-time verifier  $V_3$  for  $L_3$ . Since a certificate c for  $L_3$  will have the property that either  $V_1(x,c) = 1$  or  $V_2(x,c) = 1$ , we can easily construct a verifier  $V_3(x,c) =$  $V_1(x,c) \vee V_2(x,c)$ . Clearly then  $x \in L_3$  if and only if there is a certificate csuch that  $V_3(x,c) = 1$ . Notice also that the new verifier  $V_3$  will run in time  $O(2(|x|^d))$ , which is polynomial. Therefore, the union  $L_3$  of two languages in NP is also in NP, so NP is closed under union.

(b) Show that NP is closed under concatenation.

## Answer:

Now we will show that  $L_4 = L_1 \circ L_2$  is in NP, where  $L_1$  and  $L_2$  are languages in NP with verifiers  $V_1$  and  $V_2$  as in the solution for the previous part. Again, our goal is to construct a polynomial-time verifier  $V_4(x,c)$  for a string x and the possible certificate c. Suppose |x| = n. We can define  $V_4(x,c) = 1$  if and only if c = k # y # z, where # is a new symbol,  $k \in \{0, 1, \ldots, n\}$ , and

$$V_1(x_1 \cdots x_k, y) = 1$$
 and  $V_2(x_{k+1} \cdots x_n, z) = 1$ .

Note that k specifies the position where the original string x should be split into two parts, and y and z are the certificates for the two parts. The verifier  $V_4$ will run in time  $O(|x|^d)$  since  $|x_1 \cdots x_k| \leq |x|$  and  $|x_{k+1} \cdots x_n| \leq |x|$ . Also,  $V_4(x, w) = 1$  if and only if  $x \in L_4$ . Thus, the language  $L_4$ , the concatenation of two languages in NP, is also in NP.

2. Show that if P = NP, we can factor integers in polynomial time.

**Answer:** Define the language

$$LARGE\text{-}FACTOR = \{ \langle n, t \rangle \mid n \text{ and } t \text{ are positive integers, and } n \\ \text{has a factor } f \text{ satisfying } t \leq f < n \}.$$

We first show that LARGE- $FACTOR \in NP$ . A non-deterministic TM M that decides LARGE-FACTOR simply guesses an integer f and verifies that (a)  $t \leq f < n$  and (b) f divides n.

By invoking TM M with input  $\langle n, 2 \rangle$ , we check if n is composite or not. If n is composite, we use M to actually discover the prime factorization of n as follows:

Let us denote the largest factor of n that lies in the range (1, n) by  $f_{\text{largest}}$ . Then M accepts  $\langle n, f \rangle$  for all  $f \in [2, f_{\text{largest}}]$ , and M rejects  $\langle n, f \rangle$  for all  $f \in [f_{\text{largest}} + 1, n]$ . How do we discover  $f_{\text{largest}}$ ? If we iterate f from 1 through n, we would be doing an exponential amount of work in the size of the input (because we have O(n) iterations when the size of the input is only  $O(\log n)$ ). A faster procedure is to use binary search to identify  $f_{\text{largest}}$ . This would involve at most  $O(\log n)$  invocations of TM M.

Having discovered the largest factor of n, say  $f_{\text{largest}}$ , we compute  $p = n/f_{\text{largest}}$ , which happens to be the smallest prime factor of n. We now discover the largest factor of n/p, and so on (using the above procedure for discovering the largest factor).

Any number n can have at most  $O(\log n)$  prime factors. As we showed earlier, each prime factor requires  $O(\log n)$  invocations of TM M for its discovery. Thus, TM M is invoked  $O(\log^2 n)$  times. Under the assumption that P = NP, the whole procedure is polynomial.