MATH 4LT/6LT3 Assignment #6

Due: Monday, 1 December by 11:59pm.
Note: Assignments that are submitted after this deadline, but before 11:59pm on Friday, December 5 will be accepted, without any late penalty.

1. Exercise 3.10.15 from the textbook.

   Solution:

   (a) Suppose that the languages B and C are PTIME over the same alphabet Σ, and let $A = B \cdot C$. Let M_B and M_C be polynomial-time DTMs whose languages are B and C respectively. Let M_A be the DTM that on input a string $x = x_1 x_2 \ldots x_n \in \Sigma^*$ does the following:

   - Set $i = 0$.
   - While $i \leq |x|$,
     - Set $b = x_1 x_2 \ldots x_i$ and $c = x_{i+1} x_{i+2} \ldots x_n$,
     - Run M_B on input b and M_C on input c. If both strings are accepted, then accept x.
     - Set $i = i + 1$.
   - Reject x.

   By design, $L(M_A) = B \cdot C$. To see that M_A is a polynomial-time DTM, let $p_B(n)$ and $p_C(n)$ be polynomials that bound the runtimes of M_B and M_C respectively. We may assume that both polynomials are increasing. The main loop of the DTM M_A is executed at most $n = |x|$ times, and for each pass through the loop, the DTMs M_B and M_C are run on strings whose length is at most n. So, it will take at most $p_B(n) + p_C(n)$ steps, plus some additionalsteps to increment the counter i and set up the strings b and c. The number of these additional steps can be bounded by a (linear) polynomial $q(n)$in n. So the runtime of M_A can be bounded by $n(p_B(n) + p_C(n) + q(n))$, which is a polynomial in n. Thus $B \cdot C$ is PTIME.

(b) Suppose that the languages B and C are NP over the same alphabet Σ, and let $A = B \cdot C$. Let M_B and M_C be polynomial-time NTMs whose languages are B and C respectively. The description of the machine M_A from the previous part serves as a description of a polynomial-time NTM whose language is $B \cdot C$. The only difference is that in this part, the machines M_A and M_B are polynomial-time NTMs. In the step where these machines are run on the strings b and c, if these strings are in B and C respectively, then there will be accepting computations of M_B and M_C on these strings.

   The same polynomial bound on the runtime of M_A applies in this part as well. No matter whether the strings b and c are accepted by M_B and M_C, the computations will take at most $p_B(n)$ and $p_C(n)$ steps.

(c) Suppose that B and C are coNP languages over the alphabet Σ. To show that $B \cdot C$ is coNP, we need to show that the complement of $B \cdot C$ is NP. Rather than present a description of a polynomial-time NTM whose language is this complement, we can approach this problem in terms of verifiers. Since B and C are coNP, then their complements have verifiers. We can use them to describe a verifier for the complement of $B \cdot C$ as follows: given a string $x = x_1x_2 \dots x_n \in \Sigma^*$ to show that $x \notin B \cdot C$, we need to show that for all $i \leq n$, that either the string $x_1x_2 \dots x_i \notin B$ or that $x_{i+1}x_{i+2} \dots x_n \notin C$. So a certificate that can be used to verify that $x \notin B \cdot C$ can consist of a sequence of $n + 1$ certificates that can be used to show, for each $i \leq n$ that either $x_1x_2 \dots x_i \notin B$ or that $x_{i+1}x_{i+2} \dots x_n \notin C$. The length of this certificate can be bounded by the product of n and the polynomial bounds associated with the two verifiers for B and C.

   As an alternative to this solution, we can describe a polynomial-time NTM whose language is the complement of $B \cdot C$. The description of such an NTM is similar to the description of the DTM in part 1. The only difference is that $M_{\overline{B}}$ and $M_{\overline{C}}$ are polynomial-time NTMs whose languages are $\overline{B}$ and $\overline{C}$ respectively:

   - Set $i = 0$.
   - While $i \leq |x|$,
     - Set $b = x_1x_2 \dots x_i$ and $c = x_{i+1}x_{i+2} \dots x_n$,

- Run $M_{\overline{B}}$ on input b and $M_{\overline{C}}$ on input c. If both strings are rejected, then **reject** x.
- Set $i = i + 1$.

- Accept x.

The same estimate of the runtime for the DTM in part 1 applies to the runtime of this NTM.

2. Exercise 3.10.19 from the textbook.

   Solution: This exercise was removed from this assignment.

3. Exercise 3.10.21 from the textbook.

   Solution:

   1. Clearly $\equiv_m^p$ is reflexive, since for all B, $B \leq_m^p B$, and by definition, it is symmetric. To see that it is transitive, we can use Theorem 3.4.18 from the textbook.
   2. Let A be a nontrivial PTIME language and B any other nontrivial language. We claim that $A \leq_m^p B$. To see this, let b and b' be strings with $b \in B$ and $b' \notin B$. Define ρ to be the function such that for $x \in \{0,1\}^*$, if $x \in A$, then $rho(x) = b$ and if $x \notin A$, $rho(x) = b'$. Since A is PTIME, then the function ρ is polynomial-time computable (see Exercise 2.10.31 from Assignment #5 for a similar result). Since ρ is a many-one reduction from A to B, it follows that $A \leq_m^p B$.

      If B is a nontrivial PTIME language, then by the above $A \leq_m^p B$ and $B \leq_m^p A$ both hold, showing that $A \equiv_m^p B$. Now, suppose that $A \equiv_m^p B$. Then $Ble_m^p A$ and so by Theorem 3.4.14, it follows that B is PTIME as well. Thus the set of nontrivial PTIME languages forms an equivalence class of $\equiv_m^p$.
   3. Let A and B be NP-complete languages. Then by definition, $A \leq_m^p B$ and $B \leq_m^p A$ since both languages are also NP. Thus $A \equiv_m^p B$. On the other hand, if C is a language with $A \equiv_m^p C$, then $C \leq_m^p A$, which implies that C is an NP language. Also, $A \leq_m^p C$ implies that C is NP-hard, since the relation $\leq_m^p$ is transitive. This shows that C is also NP-complete and that the set of NP-complete languages forms an equivalence class of $\equiv_m^p$.

4. If these two equivalence classes are equal then every NP-complete language is also PTIME. Then by Theorem 3.5.5, $\mathcal{NP} = \mathcal{P}$. Also, by that theorem, we conclude that if $\mathcal{NP} = \mathcal{P}$ then each NP-complete language is PTIME, showing that the two equivalence classes are equal.

4. Let SAT_2 be the language

$$\{\ulcorner\phi\urcorner \mid \phi \text{ is a Boolean formula that has at least two satisfying assignments}\}.$$

(a) Show that SAT_2 is an NP language.

Solution: The following is a high level description of a polynomial-time NTM M such that $L(M) = SAT_2$. This establishes that SAT_2 is an NP language. Let M be the NTM that on input a string $\ulcorner\phi\urcorner$ for some Boolean formula ϕ does the following:

- Finds the set S of Boolean variables that appear in ϕ,
- guesses two different assignments $\mu : S \to \{0, 1\}$ and $\nu : S \to \{0, 1\}$,
- checks to see if both of them satisfy ϕ. If so, then M accepts the string, and if not, it rejects the string.

The second step is where the nondeterminism enters into the operation of M. The guesses of μ and ν can be carried out in time bounded by a polynomial in the length of $\ulcorner\phi\urcorner$ since the set S has size smaller than this number, and the guesses can be coded as strings of 0's and 1's of length $|S|$.

The other two steps can be carried out in polynomial time as a function of $|\ulcorner\phi\urcorner|$.

A string is accepted by M if and only if there are two distinct assignments that satisfy the formula that the string encodes. Thus $L(M) = SAT_2$.

(b) Show that $SAT \leq^p_m SAT_2$.

Solution: Given a Boolean formula ϕ, let x be some Boolean variable that does not occur in ϕ. Let ϕ' be the formula $\phi \wedge (x \vee \overline{x})$. Let ν be a truth assignment for the variables that occur in ϕ and define ν_0 and ν_1 to be truth assignments that extend ν by setting

$\nu_0(x) = 0$ and $\nu_1(x) = 1$. It is not hard to see that if ν satisfies ϕ, then both ν_0 and ν_1 satisfy ϕ'. Conversely, if μ is any truth assignment that satisfies ϕ' then μ also satisfies ϕ. From this we can conclude that ϕ is satisfiable if and only if ϕ' is satisfied by at least two assignments. We note that given the string $\ulcorner\phi\urcorner$, the string $\ulcorner\phi'\urcorner$ can be computed in polynomial-time, as a function of the length of $\ulcorner\phi\urcorner$. So, the function ρ that produces the string $\ulcorner\phi'\urcorner$ from the string $\ulcorner\phi\urcorner$ is a polynomial-time, many-one reduction from SAT to SAT_2.

(c) Is SAT_2 an NP-complete language?

Solution: Yes, since SAT is NP-complete, then by the previous part of this question we can conclude that SAT_2 is also NP-complete.

The following question is for students enrolled in MATH 6LT3. Students in MATH 4LT3 can treat it as a bonus question.

B1 Exercise 3.10.64 from the textbook.

Solution:

1. To see that PRIME-FACTOR is NP, we can show that there is a verifier for this language. At a high level, such a verifier would take as input a string of the form $\langle\langle\ulcorner m\urcorner, \ulcorner a\urcorner, \ulcorner b\urcorner\rangle, y\rangle$, with m, a, and b natural numbers and $y \in \Sigma^*$ and accept it if $y = \ulcorner p\urcorner$ for some prime number p such that p is a divisor of m and $a \leq p \leq b$. The set of strings of this form is polynomially bounded, since the length of y will be less than the length of $\ulcorner m\urcorner$. Checking whether y has the requisite form can be performed by a polynomial-time DTM whose runtime can be bounded by a polynomial in the length of the input string. Note that as part of this procedure, the fact that the language PRIMES is PTIME is used. This shows that PRIME-FACTOR is an NP language.

   To see that PRIME-FACTOR is coNP we first note that for any natural number $m > 1$, the number of prime divisors of m, counting up to multiplicity, is at most $\log_2(m) \leq |\ulcorner m\urcorner| + 1$. To see this, suppose that $m = p_1 p_2 \dots p_k$ for some prime numbers p_i and $k \geq 1$. Then

$$\log_2(m) = \sum_{i=1}^{k} \log_2(p_i) \geq k.$$

Also, since each $p_i \leq m$, it follows that the string $\langle \ulcorner p_1 \urcorner, \dots, \ulcorner p_k \urcorner \rangle$ has length that can be bounded by a polynomial in the length of $\ulcorner m \urcorner$.

We can use this to devise a verifier for the complement of PRIME-FACTOR, thereby establishing that this language is also coNP. This verifier, on input a string of the form $\langle \langle \ulcorner m \urcorner, \ulcorner a \urcorner, \ulcorner b \urcorner \rangle, y \rangle$ checks to see if y encodes a sequence of prime numbers $p_1, p_2, \dots p_k$ for some $k \geq 1$ and then checks to see if their product is equal to m. If so, then a check is made to see if any of the p_i lie between a and b. If so, the input string is accepted, and it is rejected in all other cases.

Since the length of $\langle \ulcorner p_1 \urcorner, \dots, \ulcorner p_k \urcorner \rangle$, an encoding of a prime factorization of m, can be bounded by a polynomial in the length of $\ulcorner m \urcorner$, it follows that the language of this verifier is polynomially bounded. Furthermore, since primality checking and integer multiplication can both be carried out in polynomial-time, it follows that the verifier runs in polynomial-time as well.

2. We first show that i. implies ii. Suppose that PRIME-FACTOR is PTIME. The following is a high level description of a DTM M that on input $\ulcorner m \urcorner$ outputs a string $\ulcorner d \urcorner$ where d is a proper divisor of m, if m is composite, and outputs 0 otherwise:

    - Check if m is prime or is equal to 0 or 1. If so, output 0 and halt.
    - Set $a = 1$ and $b = m$.
    - Iterate the following steps:
        - If $a = b$, then output a and halt.
        - Check to see if $\langle \ulcorner m \urcorner, \ulcorner a \urcorner, \ulcorner a + \lfloor (b-a)/2 \rfloor \urcorner \rangle$ is in PRIME-FACTOR.
        - If so, set $b = a + \lfloor (b-a)/2 \rfloor$. If not, set $a = a + \lceil (b-a)/2 \rceil$.

    If m is not prime and is greater than 1, the above algorithm starts with an interval $[a, b]$ that is guaranteed to contain some (prime) divisor of m. It then, iteratively, cuts this interval in half, and then checks to see if the lower half contains a prime divisor. If so, then the interval is reset to be this lower one. If not, it is reset to be the upper one (which is guaranteed to contain a prime divisor of m). This process continues until the interval has length 1, i.e., when $a = b$. At this point, it can be concluded that a is a prime divisor of m.

Since each pass through the loop cuts the interval being examined in half, then the number of iterations is bounded by $\log_2(m) \leq |\ulcorner m \urcorner| + 1$. So, the number of times the steps in the loop are run can be bounded by a polynomial in $|\ulcorner m \urcorner|$. Since the runtime of each step in the loop can also be bounded by a polynomial in $|\ulcorner m \urcorner|$ (since are are assuming that PRIME-FACTOR is PTIME), it follows that the DTM M runs in polynomial-time and computes the function f.

To show that ii. implies iii., suppose that the function f as described in ii. is polynomial-time computable. Let M be a DTM that does the following on input a string $\ulcorner m \urcorner$ for some natural number $m \geq 2$:

- Initialize LIST to be the string $\langle \ulcorner m \urcorner \rangle$, the code of a list of length 1.
- Iterate the following steps:
  - Search through LIST to find the first composite number.
  - If no composites are found, output LIST and halt.
  - If n is the first composite found in LIST,
    - Compute $d = f(\ulcorner n \urcorner)$. Let $q = n/d$.
    - Replace the entry $\ulcorner n \urcorner$ in LIST by the entries $\ulcorner d \urcorner$ and $\ulcorner q \urcorner$.

We claim that the above DTM computes the function f' that on input $\ulcorner m \urcorner$ outputs a prime decomposition of m, if m is a natural number greater than 1. To see this, note that at each step, the product of the numbers coded in the string LIST is equal to m, and that at the end of the running of M, the entries in LIST are all codes of prime numbers. So, upon termination, LIST contains a prime factorization of m. To see that M runs in polynomial-time, note that the length of LIST cannot exceed $|\ulcorner m \urcorner| + 1$, as noted earlier, and that at each step, the length of LIST grows by 1. So, the main loop of M is run at most $|\ulcorner m \urcorner| + 1$ times. Since the runtime of each step in the loop can be bounded by a polynomial in $|\ulcorner m \urcorner|$ (since f is assumed to be polynomial-time computable, and primality testing is also polynomial-time computable), it follows that M is a polynomial-time DTM that computes the function f'.

To show that iii. implies i., the following is a description of a polynomial-time DTM that on input $\langle \ulcorner m \urcorner, \ulcorner a \urcorner, \ulcorner b \urcorner \rangle$ does the following:

- Computes $f'(\ulcorner m \urcorner)$.
- Searches through the output of this computation to see if the list contains a prime number that lies between a and b. If so, accept the input string, and if not reject it.

Since f' is assumed to be polynomial-time computable, it follows that the above DTM runs in polynomial-time and that it correctly decides if there is a prime divisor of m that lies between a and b. So, if f' is polynomial-time computable, then PRIME-FACTOR is PTIME.