CSCE-637 Complexity Theory

Fall 2020

Instructor: Dr. Jianer Chen
Office: HRBB 338C
Phone: 845-4259
Office Hours: TR 1:30 pm–3:00 pm

Solutions to Assignment #2

1. A language L_1 is *Turing reducible* to another language L_2, written as $L_1 \leq_T^p L_2$, if there is a deterministic polynomial-time oracle Turing machine that uses L_2 as its oracle and accepts the language L_1. A language L is *NP-hard under Turing reducibility* if every language in NP is Turing reducible to L. Prove: (1) if an NP-hard language under Turing reducibility is in P, then P = NP; and (2) If a language L_1 is Karp (i.e., polynomial-time many-one) reducible to another language L_2, then L_1 is Turing reducible to L_2.

Proof. (1) Let Q be an NP-hard language under Turing reducibility. Suppose that Q is in P, i.e., Q is solvable by a deterministic Turing machine M_Q (without oracle) in time $p_Q(n)$, where $p_Q(n)$ is a polynomial of n. Now let Q' be any problem in NP. Since Q is NP-hard under Turing reducibility, there is a deterministic polynomial-time oracle Turing machine M_0 using Q as its oracle that accepts Q'. Let the running time of M_0 be bounded by a polynomial $p_0(n)$ of n.

Now consider the following deterministic Turing machine M'_Q without oracle: on an input x of length n, M'_Q simulates the oracle Turing machine M_0 on x. Whenever, M_0 writes a string y on its oracle tape and queries on y, M'_Q instead calls the Turing machine M_Q on y to determine if $y \in Q$. Once M_Q on y returns with a decision, M'_Q gets the correct answer to the query on y, and continues simulating M_0. The Turing machine M'_Q accepts x if and only if M_0 accepts x. Since every oracle query of M_0 is replaced by a call to the deterministic Turing machine M_Q (with no oracle), the Turing machine M'_Q uses no oracle. Since both M_0 and M_Q are deterministic Turing machines, the Turing machine M'_Q is also deterministic. Finally, since the running time of Turing machine M_0 is bounded by $p_0(n)$, each string y placed on the oracle tape of M_0 has its length bounded by $p_0(n)$. Thus, the running time of M_Q on y is bounded by $p_Q(p_0(n))$. Now since each step of the Turing machine M_0, including the oracle query steps, is replaced by at most $p_Q(p_0(n))$ steps in M'_Q, the running time of the Turing machine M'_Q on input x of length n is bounded by $p_0(p_Q(p_0(n)))$. Since both p_0 and p_Q are polynomials, $p_0(p_Q(p_0(n)))$ is a polynomial of n so the Turing machine M'_Q (without oracle) is a deterministic Turing machine that runs in polynomial time and accepts the language Q', i.e., Q' is in P. Since Q' is an arbitrary language in NP, this proves that NP $\subseteq$ P, leading directly to P = NP. This proves that if the NP-hard language Q under Turing reducibility is in P, then P = NP.

(2) Suppose that L_1 is Karp-reducible to L_2. By the definition, there is a function $f(x)$ computable in polynomial time such that x is in L_1 if and only if $f(x)$ is in L_2. Now construct an oracle Turing machine M_Q using L_2 as its oracle, as follows: on input x, M_Q computes $y = f(x)$ and queries if $y \in L_2$ on its oracle, M_Q accepts x if and only if the answer to the query on y is yes, which is true if and only if $y = f(x) \in L_2$, thus, if and only if $x \in L_1$. The oracle

Turing machine M_Q obviously runs in polynomial time since $f(x)$ is computable in polynomial time. Therefore, the oracle Turing machine M_Q uses L_2 as oracle, runs in polynomial time, and accepts the language L_1. That is, the language L_1 is Turing reducible to the language L_2. □

2. Define a language UNSAT $= \{F \mid F$ is an unsatisfiable CNF formula$\}$. Prove: UNSAT is NP-hard under Turing reducibility, but is unlikely to be NP-hard under Karp reducibility.

Proof. We first prove that UNSAT is NP-hard under Turing reducibility. Note that for a CNF formula F, F is a yes-instance of UNSAT if and only if F is a no-instance of SAT. Let Q be any problem in NP. Since the SAT problem is NP-complete under Karp-reduction, there is a polynomial-time computable function $f(x)$ such that x is a yes-instance of Q if and only if $f(x)$ is a yes-instance of SAT. By the definition, we can assume that $f(x)$ is a valid CNF formula. Now consider the following oracle Turing machine M_0 that uses UNSAT as oracle and solves the problem Q. On input x, M_0 first computes $f(x)$ then places $f(x)$ on its oracle tape to query if $f(x) \in$ UNSAT. The machine M_0 accepts x if and only if the oracle query to $f(x)$ returns NO. The machine M_0 runs in polynomial time since $f(x)$ is computable in polynomial time. Moreover, M_0 accepts x if and only if the query to the CNF formula $f(x)$ on the oracle UNSAT is NO, if and only if $f(x)$ is a satisfiable CNF formula, if and only if $x \in Q$. Thus, the oracle Turing machine M_0 uses UNSAT as oracle, accepts Q, and runs in polynomial time. This proves that Q is Turing-reducible to UNSAT. Since Q is an arbitrary problem in NP, this proves that UNSAT is NP-hard under Turing reducibility.

Now we prove that UNSAT is unlikely to be NP-hard under Karp-reducibility. Assuming the contrary that UNSAT is NP-hard under Karp-reducibility. Consider any co-NP problem Q. By definition, the complement $\overline{Q}$ of Q is in NP. Since UNSAT is NP-hard under Karp-reducibility, there is a polynomial-time computable function f such that x is in $\overline{Q}$, i.e., x is not in Q, if and only if $f(x)$ is in UNSAT, i.e., $f(x)$ is not in SAT. This gives that x is in Q if and only if $f(x)$ is in SAT. Now we construct the following nondeterministic algorithm M_Q to solve Q, as follow. On input x, M_Q first computes $f(x)$, then simulates the nondeterministic polynomial-time algorithm for SAT to solve $f(x)$ (remark: you should be able to construct a nondeterministic polynomial-time algorithm that solves SAT). Because x is in Q if and only if $f(x)$ is in SAT, this nondeterministic polynomial-time algorithm M_Q solves the problem Q, i.e., the problem Q is in NP. Since Q is an arbitrary problem in co-NP, this proves that co-NP $\subseteq$ NP. This also leads to NP $\subseteq$ co-NP, as follows. Let R be a problem in NP, then the complement $\overline{R}$ is in co-NP. Since co-NP $\subseteq$ NP, $\overline{R} \in$ NP. This gives $\overline{\overline{R}} = R$ is in co-NP. Thus, every problem in NP is in co-NP, and NP $\subseteq$ co-NP. In conclusion, if UNSAT is NP-hard under Karp-reducibility, then we would have NP = co-NP, which, by complexity theory, is very unlikely. □

3. Prove: the polynomial-time hierarchy PH has no complete languages under the polynomial-time reduction unless PH collapses.

Proof. Assume that the polynomial-time hierarchy PH has a complete language Q under the polynomial-time reduction. Since Q is in PH, $Q \in \Sigma_k^p$ for some fixed k. Without loss of generality, we assume $k \geq 2$. Thus, there is a nondeterministic polynomial-time oracle Turing machine M_Q that uses a language B in Σ_{k-1}^p as oracle and accepts Q.

Since Q is PH-hard under the polynomial-time reduction, for any problem R in PH, there is a polynomial-time computable function f such that x is in R if and only if $f(x)$ is in Q.

Now consider the following oracle Turing machine M_0 that uses B as oracle and accepts R: on input x, M_0 first computes $f(x)$, then simulates the nondeterministic oracle Turing machine M_Q on input $f(x)$, using oracle B. Thus, the Turing machine M_0 is also a nondeterministic oracle Turing machine. Since x is in R if and only if $f(x)$ is in Q, and since the oracle Turing machine M_Q using oracle B accepts Q, the new oracle Turing machine M_0 accepts the language R. Moreover, since the length of $f(x)$ is bounded by a polynomial of $n = |x|$, and since M_Q runs in polynomial time, the Turing machine M_0 runs in time polynomial in n. Therefore, M_0 is a nondeterministic polynomial-time oracle Turing machine that uses oracle B and accepts R. Since $B \in \Sigma_{k-1}^p$, this proves that $R \in \mathrm{NP}^{\Sigma_{k-1}^p} = \Sigma_k^p$. Since R is an arbitrary language in PH, this shows that all languages in PH are in Σ_k^p, i.e., the polynomial-time hierarchy PH collapses to Σ_k^p. This completes the proof. □

4. In the class, we showed that a problem A is in Σ_k^p if and only if A can be written as

$$A = \{x \mid \exists_{|y_1| \le p_A(|x|)} y_1 \forall_{|y_2| \le p_A(|x|)} y_2 \cdots Q_{|y_k| \le p_A(|x|)} y_k \ F_A(x, y_1, y_2, \ldots, y_k) = 1\},$$

where F_A is a polynomial-time computable Boolean function. Similarly, a problem B is in Π_k^p if and only if B can be written as

$$B = \{x \mid \forall_{|y_1| \le p_B(|x|)} y_1 \exists_{|y_2| \le p_B(|x|)} y_2 \cdots Q_{|y_k| \le p_B(|x|)} y_k \ F_B(x, y_1, y_2, \ldots, y_k) = 1\},$$

where F_B is a polynomial-time computable Boolean function.

Use these characterizations to prove that if for some $k \ge 1$, $\Sigma_k^p = \Pi_k^p$, then $\mathrm{PH} = \Sigma_k^p$.

Proof. Suppose that $\Sigma_k^p = \Pi_k^p$ for some $k \ge 1$. Consider a language A_{k+1} in Σ_{k+1}^p. By the characterization given above,

$$A_{k+1} = \{x \mid \exists_{|y_1| \le p(|x|)} y_1 \forall_{|y_2| \le p(|x|)} y_2 \cdots Q_{|y_{k+1}| \le p(|x|)} y_{k+1} \ F(x, y_1, y_2, \ldots, y_{k+1}) = 1\}, \tag{1}$$

where p is a polynomial and F is a polynomial-time computable Boolean function. Now consider the language

$$B_k = \{(x, y_1) \mid \forall_{|y_2| \le p(|x|)} y_2 \exists_{|y_3| \le p(|x|)} y_3 \cdots Q_{|y_{k+1}| \le p(|x|)} y_{k+1} \ F(x, y_1, y_2, \ldots, y_{k+1}) = 1\}. \tag{2}$$

Starting with a $\forall$ quantifier, there are k quantifier alternations in the expression for B_k. Thus, $B_k \in \Pi_k^p$. By the assumption $\Sigma_k^p = \Pi_k^p$, we have $B_k \in \Sigma_k^p$. Thus, B_k can also be written as

$$B_k = \{(x, y_1) \mid \exists_{|y_2| \le p'(|(x,y_1)|)} y_2 \forall_{|y_3| \le p'(|(x,y_1)|)} y_3 \cdots Q_{|y_{k+1}| \le p'(|(x,y_1)|)} y_{k+1} \ F'(x, y_1, y_2, \ldots, y_{k+1}) = 1\},$$

where p' is a polynomial and F' is a polynomial-time computable Boolean function. Since $|y_1| \le p(|x|)$, $p'(|(x, y_1)|)$ is bounded by a polynomial p_1 of $|x|$. Thus, the condition "$\le p'(|(x, y_1)|)$" can be replaced by "$\le p_1(|x|)$", and B_k can be re-written as

$$B_k = \{(x, y_1) \mid \exists_{|y_2| \le p_1(|x|)} y_2 \forall_{|y_3| \le p_1(|x|)} y_3 \cdots Q_{|y_{k+1}| \le p_1(|x|)} y_{k+1} \ F'(x, y_1, y_2, \ldots, y_{k+1}) = 1\}. \tag{3}$$

From (1) and (2), we can re-write the language A_{k+1} as

$$A_{k+1} = \{x \mid \exists_{|y_1| \le p(|x|)} y_1 \ (x, y_1) \in B_k\}. \tag{4}$$

Bringing the expression of B_k in (3) into (4), we get

$$\begin{aligned} A_{k+1} &= \{x \mid \exists_{|y_1|\le p(|x|)} y_1 \exists_{|y_2|\le p_1(|x|)} y_2 \forall_{|y_3|\le p_1(|x|)} y_3 \cdots Q_{|y_{k+1}|\le p_1(|x|)} y_{k+1} \\ &\qquad F'(x, y_1, y_2, \ldots, y_{k+1}) = 1\} \\ &= \{x \mid \exists_{|(y_1,y_2)|\le p_2(|x|)} (y_1, y_2) \forall_{|y_3|\le p_2(|x|)} y_3 \cdots Q_{|y_{k+1}|\le p_2(|x|)} y_{k+1} \\ &\qquad F''(x, y_1, y_2, \ldots, y_{k+1}) = 1\}, \quad (5) \end{aligned}$$

where $p_2(n) = p(n) + p_1(n)$ is a polynomial of $n = |x|$, and F'' is a trivial modification of F' such that $F''(x, y_1, y_2, \ldots, y_{k+1}) = 0$ if $|y_1| > p(|x|)$, or $|y_i| > p_1(|x|)$ for any $i > 1$, — otherwise $F''(x, y_1, y_2, \ldots, y_{k+1}) = F'(x, y_1, y_2, \ldots, y_{k+1})$. The Boolean function F'' is polynomial-time computable since the Boolean function F' is polynomial-time computable.

By (5), the language A_{k+1} can be written as a quantified expression with k alternations, starting with the quantifier $\exists$, with p_2 being a polynomial and F'' being a polynomial-time computable Boolean function. By the characterization, A_{k+1} is in Σ_k^p. Since A_{k+1} is an arbitrary language in Σ_{k+1}^p, this proves $\Sigma_{k+1}^p \subseteq \Sigma_k^p$, i.e., $\Sigma_{k+1}^p = \Sigma_k^p$.

The rest is a routine derivation. Assume inductively, $\Sigma_{k+h}^p = \Sigma_k^p$ for $h > 0$. This holds true for $h = 1$ as shown above. Now consider $\Sigma_{k+h+1}^p = \mathrm{NP}^{\Sigma_{k+h}^p}$. By induction, $\Sigma_{k+h}^p = \Sigma_k^p$, thus,

$$\Sigma_{k+h+1}^p = \mathrm{NP}^{\Sigma_k^p} = \Sigma_{k+1}^p = \Sigma_k^p,$$

and the induction goes through. This proves that $\Sigma_{k+h}^p = \Sigma_k^p$ for all $h > 0$, i.e., $\mathrm{PH} = \Sigma_k^p$ and the polynomial-time hierarchy PH collapses to the k-th level Σ_k^p. $\square$