CSE596 Problem Set 8 Answer Key Fall 2012

(1) A collection $\{\mathcal{C}_i\}$ of complexity classes forms a *proper hierarchy* if given any $\mathcal{C}_i$ and $\mathcal{C}_j$ with $i \neq j$, one of them is properly contained in the other. Which of the following collections are proper hierarchies? Justify your answers, mainly by verifying the relevant "little-o" or "Θ" relations between time bounds. Here $\mathbf{Q}^+$ stands for the positive rational numbers.

(a) $\{\,\mathrm{DTIME}[n^c] : c \in \mathbf{Q}^+, c \geq 1\,\}$.

(b) $\{\,\mathrm{DTIME}[(n+c)^3] : c \in \mathbf{Q}^+, c \geq 1\,\}$.

(c) $\{\,\mathrm{DTIME}[2^{cn}] : c \in \mathbf{Q}^+\,\}$.

(d) $\{\,\mathrm{DTIME}[2^{n^{1/c}}] : c \in \mathbf{Q}^+, c \geq 1\,\}$.

(24 pts. total, plus 15 pts. extra credit if you do (e) $\{\,\mathrm{DTIME}[n^2(\log n)^{1/c}] : c \in \mathbf{Q}^+, c \geq 1\,\}$.)

Regular-Credit Answers:

All of the functions involved are time constructible, since c and d are rational. Hence we need only check, for each case of a time bound $t(n,c)$ and given $c < d$, whether $t(n,c)\log t(n,c) = o(t(n,d))$, i.e. whether $t(n,c)\log t(n,c)/t(n,d) \longrightarrow 0$ as $n \longrightarrow \infty$. For convenience we assume that logs are to base e; in none of these cases does the base of the logarithms matter.

(a) $\{\,\mathrm{DTIME}[n^c] : c \in \mathbf{Q}^+, c \geq 1\,\}$: We get $n^c \log(n^c)/n^d = c(\log n)/n^{d-c}$. One application of L'Hôpital's Rule turns the ratio into $(c'/n)/(d-c)n^{d-c-1}$, where $c' = c$ give-or-take a factor of $\log_2 e$ I-forget-which, and this equals $(c'/(d-c))$ times $1/(n \cdot n^{d-c-1})$. Ignoring the constants leaves $1/n^{d-c}$, which goes to 0 since $d > c$. So this is a proper hierarchy.

(b) $\{\,\mathrm{DTIME}[(n+c)^3] : c \in \mathbf{Q}^+, c \geq 1\,\}$: For all c and d, $(n+c)^3$ and $(n+d)^3$ are both $\Theta(n^3)$, since the highest-powered term is n^3. By the "Linear Speed-Up Theorem," all of these classes are equal, so there is no proper hierarchy.

(c) $\{\,\mathrm{DTIME}[2^{cn}] : c \in \mathbf{Q}^+\,\}$: We get $2^{cn} \cdot cn/2^{dn} = cn/2^{(d-c)n}$. A trip to the L'Hôpital eliminates the numerator and leaves the denominator $1/2^{(d-c)n}$ times some constants, and again this $\longrightarrow 0$ as $n \longrightarrow \infty$. So this is a proper hierarchy.

(d) $\{\,\mathrm{DTIME}[2^{n^{1/c}}] : c \in \mathbf{Q}^+, c \geq 1\,\}$: Note that here higher d makes the time bound smaller, so we get $2^{n^{1/d}} n^{1/d}/2^{n^{1/c}} = n^{1/d}/2^{n^{1/c} - n^{1/d}}$. L'Hôpital gets messy here, but a permissible handwave is to reason that for all $d > c$, there exists an n_0 such that for all $n > n_0$, $n^{1/d} < n^{1/c}/2$. For such n the ratio is bounded above by $n^{1/d}/2^{n^{1/c}/2}$. Since $d > 1$ this is certainly also bounded above by $n/2^{n^{1/c}/2}$. Now L'Hôpital kills the numerator like in (c), and the denominator stays something that clearly goes to ∞ as n does. Hence this bounding ration goes to 0, taking the original ratio down with it. Thus this is a proper hierarchy.

(2) Let $\mathcal{C}$ and $\mathcal{D}$ be any two language classes that have complete sets and are closed downward under $\leq_m^p$. Which of the following must be true? Prove those that are true, and give concrete counterexamples for those that aren't. (18 pts. total)

(a) $\mathcal{C} \cap \mathcal{D}$ is closed downward under $\leq_m^p$.

(b) $\mathcal{C} \cap \mathcal{D}$ has a complete set under $\leq_m^p$.

(c) $\mathcal{C} \cup \mathcal{D}$ is closed downward under $\leq_m^p$.

(d) $\mathcal{C} \cup \mathcal{D}$ has a complete set under $\leq_m^p$ if and only if $\mathcal{C} = \mathcal{D}$.

(a) $\mathcal{C} \cap \mathcal{D}$ is closed downward under $\leq_m^p$: *True.* Given $A \leq_m^p B$ and $B \in \mathcal{C} \cap \mathcal{D}$, the downward closure of $\mathcal{C}$ implies $A \in \mathcal{C}$, and that of $\mathcal{D}$ implies $A \in \mathcal{D}$, so $A \in \mathcal{C} \cap \mathcal{D}$.

(b) $\mathcal{C} \cap \mathcal{D}$ has a complete set under $\leq_m^p$: *False.* The simplest (hinted-at) counterexample is to take $\mathcal{C} =$ RE and $\mathcal{D} =$ co-RE. Then $\mathcal{C} \cap \mathcal{D} =$ REC, but the class of recursive languages has no complete sets under $\leq_m^p$. The *reason* is that if REC had a complete set B under $\leq_m^p$, then we would get a decidable set S of total Turing machines accepting all recursive languages. [Proving this is a good self-study exercise; I shortchanged the text's coverage of *recursive enumerations* in my lectures.] But the diagonal set D_S for S would be decidable, which is a contradiction.

(c) $\mathcal{C} \cup \mathcal{D}$ is closed downward under $\leq_m^p$: *True.* Given $A \leq_m^p B$ and $B \in \mathcal{C} \cup \mathcal{D}$, either $B \in \mathcal{C}$, in which case $A \in \mathcal{C}$, or $B \in \mathcal{D}$, in which case $A \in \mathcal{D}$. Either way, $A \in \mathcal{C} \cup \mathcal{D}$.

(d) $\mathcal{C} \cup \mathcal{D}$ has a complete set under $\leq_m^p$ if and only if $\mathcal{C} = \mathcal{D}$: *False*. A counterexample is $\mathcal{C} =$ RE and $\mathcal{D} =$ P. The complete set for RE shown in class is complete for $\mathcal{C} \cup \mathcal{D}$, but $\mathcal{C} \neq \mathcal{D}$. Without being so drastic as to go up to RE, one can take $\mathcal{C} =$ EXP instead, since EXP has complete sets and is closed downward under $\leq_m^p$ (self-study exercise).

(3) Prove that NP = co-NP if and only if for some NP-complete language A, its complement $\tilde{A}$ also belongs to NP. (12 pts.)

Answer: If NP = co-NP, then all languages in NP have their complement in NP, so this certainly holds for "some NP-complete set." If some NP-complete set B has its complement $\tilde{B}$ in NP, then let any language $A \in$ NP be given. Since $A \leq_m^p B$, $\tilde{A} \leq_m^p \tilde{B}$. By $\tilde{B} \in$ NP, that reduction gives us $\tilde{A} \in$ NP, so $A \in$ co-NP. Since A was arbitrary in NP that gives NP $\subseteq$ co-NP, and that implies NP = co-NP.

(5.2) **Extra Credit—optional** Prove that for all languages A, $\mathrm{NP}^A = \mathrm{NP} \iff A \in \mathrm{NP} \cap \text{co-NP}$. (24 pts.)

First suppose $\mathrm{NP}^A = \mathrm{NP}$. Since A and $\tilde{A}$ always belong to P^A, they belong to NP^A, hence to NP. But $A, \tilde{A} \in$ NP is equivalent to $A \in \mathrm{NP} \cap$ co-NP.

The converse part has the key idea of both the proof of the Arithmetical Hierarchy Theorem (Theorem 4.1 in my notes on that topic) and the analogue for the Polynomial Hierarchy (Theorem 7.11 in the text). The idea is to separate those queries answered "yes" and those answered "no" as declared in the computation being traced, and use different logic on them.

Suppose $A \in \text{NP} \cap \text{co-NP}$. Then there are polynomials q, r and polynomial-time decidable predicates Q and R such that for all x,

$$\begin{aligned} x \in A &\iff (\exists y : |y| \leq q(|x|))\ Q(x, y), \text{ and} \\ x \notin A &\iff (\exists z : |z| \leq r(|x|))\ R(x, z). \end{aligned}$$

Now let $L \in \text{NP}^A$; we have to show $L \in \text{NP}$. Take a nondeterministic oracle TM N that accepts L with oracle A in some polynomial time $p(n)$. Now we define a nondeterministic *non-oracle* TM N' that on any input w simulates the computation $N^A(w)$. Whenever N^A writes a query string x, N' guesses either a y such that $Q(x, y)$ or a z such that $R(x, z)$. Note that the existence of y or z is mutually exclusive, so a successful guess by N' really does return the correct oracle answer "yes" or "no" to the query "is $x \in A$?" Thus N' accepts w iff it successfully traces out an accepting computation of N^A on w, and such a trace always exists when $w \in L$. Hence $L = L(N^A) = L(N')$. Since the total time for the trace is bounded by $p(n) * (q(p(n) + r(p(n)))$, which is still a polynomial, N' is an NP-machine, so $L \in \text{NP}$. Since L was an arbitrary member of NP^A, we have $\text{NP}^A \subseteq \text{NP}$, so they are equal.

(4) Homer-Selman, exercise 6.17 on page 142 (page 147 in older editions). (12 pts.)

Answer. Clearly this language (call it SAT2) is in NP, with the witness being *two* satisfying assignments whenever ϕ belongs to L. To show it is NP-complete, we need to reduce 3SAT to it. This means finding a polynomial-time computable function f that alters any given 3CNF formula into a formula ϕ' such that

ϕ has at least one satisfying assignment $\iff$ ϕ' has at least two satisfying assignments .

Our strategy can be to add one "extraneous" satisfying assignment while preserving any others that ϕ has.

The easiest way to do this takes advantage of neither the definition of L in the text nor correctness of the reduction requires ϕ' to be a 3SAT formula. We can make ϕ' a 4SAT formula: First define ϕ_0 by adding a "dummy variable" c to every clause. Then the "extraneous" satisfying assignment a will involve setting c true, while setting c false leaves us with only the satisfying assignments we originally had. To make this idea work technically, we first need to create a formula ψ that has exactly one satisfying assignment when $c = 1$, and exactly one when $c = 0$. Such a formula is simply $(c \vee d) \wedge (\bar{c} \vee \bar{d})$. *However*, when we trivially satisfy the original clauses of ϕ by choosing $c = 1$, we need to prevent the original variables $x_1, \ldots, x_n$ of ϕ from compounding the resulting number of satisfying assignments. This can be done by forcing all those variables to be *true* via

$$\psi = (c \vee d) \wedge (\bar{c} \vee \bar{d}) \wedge (d \vee x_1) \wedge (d \vee x_2) \wedge \ldots \wedge (d \vee x_n).$$

The reason is that when $c = 1$, d must be 0 (i.e., false) by the second clause of ψ, and thus every x_i must be set true in that assignment. When $c = 0$, d must be 1, and that sets $x_1, \ldots, x_n$ at complete liberty to be anything they want in the original formula. Then $f(\phi) = \phi' = \phi_0 \wedge \psi$ can be our "Final Answer" and is clearly computable in polynomial time (indeed, linear-time via one L-to-R pass over ϕ). (If for purely aesthetic reasons you want ψ too to be a 4CNF formula,... that's a self-study exercise. In the next problem it is important that not only ϕ' but also the "extraneous" satisfying assignment a are computed in polynomial time.)

(5) Apply ideas of problem (4) to create a polynomial-time computable function $h : \Sigma^* \longrightarrow \Sigma^*$ for which

$$L_h = \{ x : (\exists y \neq x) h(y) = h(x) \}$$

is NP-complete. Your h will not (cannot!?) be exactly length-preserving as on Prelim II, but it must be *polynomially honest*: there must be a polynomial $q(n)$ such that for all x, $q(|h(x)|) \geq |x|$. Hint: make satisfying assignments cause collisions. (24 pts.)

Answer: Given a formula ϕ and an assignment a, define $h(\langle \phi, a \rangle) = \phi$ if $\phi(a) = 1$, and $h(\langle \phi, a \rangle) = \phi \wedge \alpha_a$ otherwise. Here $\alpha_a = x_1^{a_1} \wedge \cdots \wedge x_n^{a_n}$, where for each i, $x_i^1 = x_i$ and $x_i^0 = \bar{x}_i$. Thus in the latter case, $\phi \wedge \alpha_a$ preserves both ϕ and the information in a, so nothing else in the domain of h can ever collide with it. Whereas, when $\phi(a) = 1$, $h(\langle \phi, a \rangle) = \phi$ generates a collision with $h(\langle \phi, b \rangle)$ whenever $b \neq a$ and $\phi(b) = 1$, i.e. when ϕ has more than one satisfying assignment. Hence we have

$$L_h = \{ \langle \phi, a \rangle : \phi(a) = 1 \ \wedge \ \phi \in \text{SAT2} \}$$

and it follows that modifying the reduction f from Problem (1) to include the "extraneous assignment a" as well gives a reduction from 3SAT to L_h.

The function h is clearly polynomial-time computable. Moreover, since always $|a| < |\phi|$, $|h(\langle \phi, a \rangle)| \geq |\langle \phi, a \rangle|/2$, so h is in fact *linearly* honest. Thus $L_h \in$ NP, so it is NP-complete.

Extra Credit Answer: For *Halves*(L), consider pairs of the form $\langle \phi, x \rangle$ where x is a truth assignment. The language L of these pairs belongs to P. But, as is especially clear when the "pairing function" is just to concatenate as $\phi \# x$, the instances of *Halves*(L) include cases where you are given just the ϕ part, and boil down to asking whether there exists an x that satisfies ϕ, which is SAT and so is equally NP-complete.

(6) Show that the following decision problem is NP-complete:

DOMINATING SET
INSTANCE: An undirected graph $G = (V, E)$ and an integer k, $1 \leq k \leq |V|$.
QUESTION: Does there exist a subset $U \subseteq V$ of size at most k such that every other vertex in V is adjacent to one in U (that is, $(\forall v \in V \setminus U)(\exists u \in U) : (u, v) \in E$)?

Answer: This one is almost the simplest example of the "ladder/clause-gadget" architecture of a reduction from 3SAT that I know. Given ϕ with n variables and m clauses, set $k = n$ and G with $3n + m$ nodes as follows: For each variable x_i, in place of a simple "rung" connecting x_i to its opposite $\bar{x}_i$ we add one more node t_i connected to both x_i and $\bar{x}_i$ in a triangle. And G has just one node c_j for each clause C_j in ϕ. Node c_j is connected to the (up to) 3 literals that belong to the clause C_j. That finishes the description of what is clearly a polynomial time (in fact, linear time and log space) computable function $f(\phi) = \langle G, n \rangle$.

For correctness, note that since each t_i is connected only to x_i and $\bar{x}_i$, there is never any reason to prefer choosing it when either x_i and $\bar{x}_i$ will dominate the triangle equally well. No node can dominate more than one t_i, so the minimum possible size for a dominating set in G is n. Thus without loss of generality, a size-n dominating set U contains exactly one of x_i and $\bar{x}_i$ for each i. These choices correspond to a truth assignment. The set U needs to dominate all nodes c_j as well, and this happens if and only if the assignment U represents satisfies ϕ. Hence f reduces 3SAT to DOMINATING SET.