CSE596 **Final Exam** **Dec. 12, 2016**

Open book, open notes, closed neighbors, 170 minutes. The exam has seven problems—the last a brief essay—and totals 267 pts., subdivided as shown. *Show your work*—this may help for partial credit. *Please write in the exam books only* (if you enclose separate scratch sheets, they may—or may not—be ignored).

Notation: As always $\mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$, while $\mathsf{EXP} = \mathsf{DTIME}[2^{n^{O(1)}}]$, and the names of other complexity classes are either completely standard or explained in the questions themselves. The alphabet Σ over which languages are encoded is immaterial; you are always welcome to consider $\Sigma = \{0,1\}$ or $\Sigma = \{0,1,\#\}$. The alphabet Γ used as the worktape alphabet of Turing machines may, however, be much larger. The tupling notation $\langle x,y\rangle$ or $\langle x,y,z\rangle$ may be considered either as representing the strings $x\#y$ and $x\#y\#z$ or as the application of pairing functions as in the Arora-Barak text; in no problem does the difference really matter.

You may cite any major relevant theorem or fact or definition covered in the course without needing to give a justification (unless one is specifically asked for). The first two questions refer to "current knowledge" in complexity theory.

(1) (50 pts.)

Classify each of the following languages $L_1, \dots, L_{10}$ according to whether it is currently known to be

(a) in deterministic logspace;

(b) in P and not known to be—or believed not to be—in deterministic logspace;

(c) in NP and strongly believed not to belong to $\mathsf{co\text{-}NP}$;

(d) in $\mathsf{co\text{-}NP}$ and strongly believed not to belong to NP;

(e) in EXP and definitely known not to be in P.

(f) recursive but known to be not in EXP;

(g) c.e. but not recursive;

(h) co-c.e. but not c.e.; or

(i) neither c.e. nor co-c.e.

There is a unique best answer for each language. Below, "M," "M_1," and "M_2" stand for deterministic Turing machines that accept languages, including the case of M being a DFA in L_9, while in L_7 "T" stands for a Turing machine that computes a partial function from $\mathbf{N}$ to $\mathbf{N}$. In L_3, "n" stands for a natural number in standard binary notation—as opposed to the unary notation of 0^n in L_4. None of this shorthand is unusual—you could infer it from context even without this note. The language(s) for (e) are in fact complete for EXP under $\leq_m^p$. The languages are (overleaf):

1. $L_1 = \{ M : L(M) \text{ is infinite} \}$.
2. $L_2 = \{ \langle M, x \rangle : M \text{ accepts } x \}$.
3. $L_3 = \{ \langle M, x, n \rangle : M \text{ accepts } x \text{ within } n \text{ steps} \}$.
4. $L_4 = \{ \langle M, x, 0^n \rangle : M \text{ accepts } x \text{ within } n \text{ steps} \}$.
5. $L_5 = \{\langle M_1, M_2, 0^n \rangle$: for all inputs y such that $M_1(y)$ and $M_2(y)$ halt within n steps, they either both accept y or both reject $y\}$.
6. $L_6 = \{\langle G, k \rangle$: there are k vertices in the undirected graph G, every two of which are connected by an edge$\}$.
7. $L_7 = \{T$: there does not exist an x such that $T(x)$ outputs 7$\}$.
8. $L_8 = \{z : z$ is the code of a Turing machine M_z, such that this code transparently includes an integer k and a time clock that shuts off computations within 2^{n^k} steps on inputs of any length n, and such that M_z does not accept $z\}$.
9. $L_9 = \{\langle M, x \rangle : M$ is a deterministic finite automaton, and M does not accept $x\}$.
10. $L_{10} = \{A : p$ is a polynomial, and if its domain is restricted to the domain $\{ 0, 1 \}^n$, then its range includes 1$\}$.

Please write your answers in this form: if L_{11} were the language of the Halting Problem, you would write "11. g" or "11. (g)". *No justifications are needed*, but may help for partial credit.

(2) 30 pts. total

Below are four statements about languages and complexity classes that are commonly disbelieved but are currently unknown to be false. For each one, write down as many **other** *currently-unknown* relations among complexity classes and languages that would follow. The other classes you may need to reference include L (i.e., deterministic logspace, also called DLOG), NL, DSPACE$[(\log n)^2]$, P, NP, co-NP, DLBA (i.e., deterministic linear space), NLBA, PSPACE, E, NE, and EXP. The languages/problems to consider are SAT, TAUT, GAP, QBF, and FACTORING. The relations to consider are equality ($=$), inequality ($\neq$), containments ($\subseteq$, $\supseteq$), proper containments ($\subset$, $\supset$), and membership or non-membership of particular languages in a class.

(a) GAP $\in$ L.

(b) TAUT $\in$ P.

(c) NLBA $\subseteq$ NP.

(d) BQP $=$ NL.

Ten correct and not-obviously-redundant relations spread among (a)–(d) suffice for full credit, but you are welcome to suggest more.

(3) (78 pts. total)

A *finite-state transducer* (FST), which by default is deterministic, is a DFA that can produce output. Formally an FST T is defined as $(Q, \Sigma, \Gamma, \rho, s, \phi)$ where:

- Q, Σ, and s are as with a DFA.
- $\rho \subseteq Q \times \Sigma \times \Gamma^* \times Q$ incorporates output using the *output alphabet* Γ as well as the transition. An instruction $(p, c/u, q)$ means the machine in state p reading $c \in \Sigma$ outputs the string $u \in \Gamma^*$ (which could be the empty string) and goes to state q. (Note that although u can be any string, each machine has a maximum length of u.)
- $\phi : Q \longrightarrow \Gamma^* \cup \{\perp\}$ provides "final output" when the machine halts. The idea of a *rejecting* state q is replaced by having $\phi(q) = \perp$ which intuitively *cancels* all previous output and can propagate rejection to another machine M that might be employing T.

A meaningful example is an FST with two states (e, o) for "even" and "odd" and $\Sigma = \Gamma = \{0, 1\}$ that carries out a "parity check." It has

$$\rho = \{(e, 0/0, e), (e, 1/1, o), (o, 0/0, o), (o, 1.1, e)\},\ \phi(e) = 0,\ \phi(o) = 1.$$

It copies the input to the output except that at the end, if the number of 1s was odd it appends a 1 to make it even, else a 0. If we wanted to mandate that the number of 1s is even to begin with, we could define $\phi(o) = \perp$ instead—which is actually what used to happen when old IBM PCs showed a "PARITY CHECK" error on a green screen of death.

A simpler example T has just one state s and uses $(s, 0/00, s)$ and $(s, 1/11, s)$ with $\phi(s) = \lambda$; it just doubles each bit, e.g., $T(010) = 001100$. But we will start off with the idea of the FST T_{Bal} which has $\Sigma = \Gamma = \{(,)\}$ and does the following:

- If the first bit is ')' then go to a dead state that will eventually output $\perp$.
- Else, take the next bits in pairs. If the pair is '((' output just one '('; if it is '))' output ')'; if it is '()' or ')(' output λ.
- If there isn't a single ')' left over at the end, hit the $\perp$ "`panic` button" like before.

The idea is that a parenthesis string x is balanced if and only if $x' = T_{Bal}(x)$ doesn't bomb and is balanced—and the length of x' is less than half the length of x. Implicit here is that $T_{Bal}(\lambda) = \lambda$ and λ counts as balanced—the start state s has $\phi(s) = \lambda$ rather than bomb. You are *not* asked to prove these facts, but they supply motivation.

(a) Write out full instruction-level code for T_{Bal}. An arc-node diagram like for a Turing machine is fine. (18 pts.)

(b) Separately, show that the language *Bal* of balanced-parenthesis strings belongs to L (that is, deterministic logspace). As before, a memory-map sketch of a Turing machine is fine, *not* necessarily a full arc-node diagram. (18 pts.)

(c) Now picture T_{Bal} as running on its own output stream, iterating until it either bombs and rejects or ends up with λ and accepts. For example, on $x = (\ ((\ )(\ ((\ ))\ ))\ )$, the first iteration will give $x' = (())$ and then $T_{Bal}(x') = \lambda$ so the overall run accepts. Prove that the language of this process is in L by a different idea, namely that one can track each of the $O(\log n)$ levels of recursion in one left-to-right pass, handling output of the next stream while the first one is still going.

For max credit, your proof should be general enough that it applies to the language defined by iterating any "length-halving" FST down to λ, not so specific to T_{Bal}. (18 pts.)

(d) Now let's forget about iterating and stick with one pass by the FST. Let's also add a second kind of bracket: [,] to go with (,), so $\Gamma' = \{(,),[,]\}$. Define Bal_2 to be the language of balanced strings that can mix these brackets. Build an FST T' that operates on strings of the form $x = u\#v$ where $u, v \in \{0,1\}^*$ (you can make it bomb if it doesn't see exactly one # char), such that x is a marked palindrome (i.e., $u = v^R$) if and only if $T'(x) \in Bal_2$. (12 pts.)

(e) What goes wrong if you don't have the # character—that is, if you want x to be a palindrome iff $T'(x) \in Bal_2$? (6 pts.)

(f) Finally, if R is a regular language, T is an FST, and L is a language such that for all strings x,

$$x \in L \iff T(x) \in R,$$

must L be regular? Justify your answer briefly. (6 pts., for 78 total on this big problem.)

(4) (12 + 30 = 42 pts.)

Write a formal mathematical definition of the language of the following decision problem, using standard graph notation. Then show that the problem is NP-complete.

PERFECT DOMINATING SET
INSTANCE: An undirected graph G, an integer $k \geq 1$.
QUESTION: Is there a set S of at most k nodes such that every other node is adjacent to *exactly* one node in S?

Note: This is not the same as the standard DOMINATING SET problem. In a 3-node triangle graph, any one node forms a perfect dominating set, but the 5-node pentagon graph does not have any perfect dominating set of size 2.

(5) (24 pts.) *True-False with reasons.*

For each statement (a)–(d), write out `true` or `false` (3 pts.), and then write a *brief* justification (3 pts.).

(a) It is known that the problem of factoring integers belongs to P if and only if $\mathsf{NP} = \mathsf{P}$.

(b) NP has complete languages under polynomial-time many-one reductions ($\leq_m^p$), but does not have any complete languages under log-space many-one reductions ($\leq_m^{\log}$).

(c) The intersection of two languages in NP always belongs to NP.

(d) If a language A is not in P, then for every Turing machine M such that $L(M) = A$, and every $x \in \Sigma^*$, the computation $M(x)$ takes an exponential number of steps on the instance x.

(6) (24 pts.)

Anti-virus utilities for personal computers work by maintaining a database of "tell-tale substrings" of *known* viruses, but are often ineffective against new, *unknown* viruses. A better, "perfect" kind of anti-virus utility would be able to analyze any downloaded software program P and determine whether running P would unleash a virus or not. Using a reduction, explain why no such "perfect" anti-virus utility can ever exist. (You do *not* need a technical definition of a PC virus to answer this question!)

(7) (19 pts.) ***Short essay answer.***

Most critics of the Church-Turing thesis have argued that it is too strong—i.e. imposes too strong a limitation on human thought process that they contend cannot be modeled by a Turing machine. (The famous physicist Roger Penrose is one of them, in his books *The Emperor's New Mind* and *Shadows of the Mind*.) *Argue the opposite*—argue that the model corresponding to human cognition should be *weaker* than the general Turing machine, such as a Turing machine restricted in its manner of operation or in allotted computational resources. What if, like Penrose, you think the brain uses quantum processes?

Write a brief discussion of major points relevant to the topic. Three major and relevant points or examples will suffice for full credit. Your answer should fit within two exam book pages—there is no need to drag it out longer once you show a strong understanding of the issues.

END OF EXAM.