UVa - cs302: Theory of Computation Spring 2008

Final Exam Comments

	< 50	50–59	60–69	70–79	80–89	90–94	95-102
Total	2	6	8	22	16	16	12

Problem 1: Short Answers. (20) For each question, provide a correct, clear, precise, and concise answer from the perspective of a theoretical computer scientist.

a. [5] What is a *language*?

Answer: A set of strings.

b. [5] Describe a regular language that *cannot* be recognized by any current laptop.

Answer: Any infinite regular language includes strings that are longer than can be stored in any real computer. So, for example, $A = (01)^*$. This assumes the entire input must be stored; if it is streamed, then it can be recognized. So, a better example would be a regular language that requires more states than can be stored on a current laptop. A language consisting of a set of 10^{1000} random strings would qualify.

c. [5] Explain why NP *cannot* stand for *non-polynomial* (even if P $\neq$ NP).

Answer: The class NP *includes* the class P, so it includes all polynomial time problems.

d. [5] Explain the basic structure of a proof that language A is NP-Complete.

Answer:

1. Prove that A is in NP, usually by showing that there is a polynomial-time verifiable certificate for A.

2. Prove that A is NP-Hard, by showing that a problem B that is known to be NP-Complete can be reduced to A.

Problem 2: Zero, One, Infinity. (14)

For each question, circle **0**, **1** or ∞ to indicate wether the value of the described entity is zero, one, or infinite. A correct answer receives full credit without any explanation. A wrong answer with a good explanation may receive some partial credit.

a. [2] Assuming P = NP, the number of NP-Complete problems that are not in P.

Answer: 0. If P = NP, all NP-Complete problems are in P.

b. [2] Assuming P = NP, the number of NP-Hard problems that are not in P.

Answer: ∞. NP-Hard includes everything outside the NP-Complete inner circle.

c. [2] The number of strings that are in the language described by the context-free grammar G (S is the start variable):

$S \to 0A$
$S \to 1S$
$A \to S$

Answer: 0. There is no way to produce a string, since all of the productions include variables on the right side.

d. [2] The number of strings that are in the language described by the context-free grammar G (S is the start variable):

$S \to 0A$
$S \to 1$
$A \to S$

Answer: ∞. The language described is 00*1.

e. [2] The smallest possible number of strings in a language that is undecidable.

Answer: 1. If it contains 0 strings, then it can't be undecidable (since all inputs reject). An example of an undecidable language that contains one string is:

f. [2] The number of languages that are equivalent to $HALT_{TM}$.

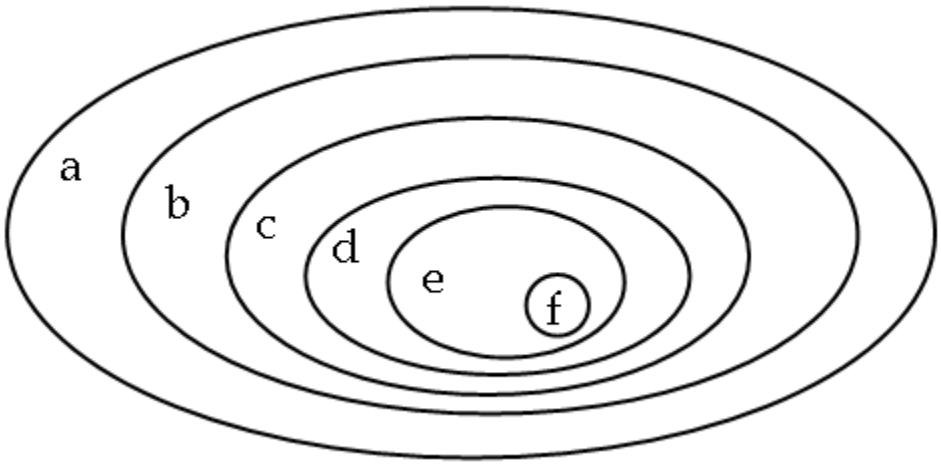
Answer: 1. (Recall question 1a: a language is a set of strings.)

g. [2] The number of languages that are in **TIME**(N^2) that can be decided by a multi-dimensional Turing machine in $O(N)$ steps.

Answer: ∞. **TIME**(N^2) includes all languages that can be decided by a regular TM in $O(N)$ steps, so obviously, these languages can be decided by a MDTM on $O(N)$ steps too.

Problem 3: Drawing Classes. (16)

Each label in the diagram below corresponds to one of the following computability and complexity classes: (1) **TIME**(N^2), (2) P, (3) Decidable, (4) NP, (5) **TIME**(1), (6) Regular.



Assuming P $\neq$ NP, identify the class (1-6) associated with each label (circle the correct answer):

a. (3) Decidable
b. (4) NP
c. (2) P
d. (1) **TIME**(N^2)
e. (6) Regular
f. (5) **TIME**(1)

g. Which of the given classes include the language $\{0^i 1^j | j > i\}$? (circle *all* classes that include this language)

(1) **TIME**(N^2) (2) P (3) Decidable (4) NP

Answer: The given language is non-regular, but can be recognized by a deterministic pushdown automaton. This means it is certainly Decidable, within NP, and within P, since we can simulate a PDA using a Turing machine in polynomial time. Deciding if it is also in **TIME(**N^2**)** is a bit more difficult, since we need to think about the number of steps required to decide it. In general, we can simulate a PDA with a TM using $O(N^2)$ steps since we can simulate the PDA with the TM by putting the stack after the input. Since the PDA is a deterministic PDA, each step consumes one input, and adds or removes at most one symbol from the stack. So, there are N steps. The maximum work for each step is to move to the square that represents the top of the stack (the rightmost square), read the stack symbol, and write a new stack symbol (overwriting the blank to the right of the top of the stack). The maximum size of the stack is N symbols (since it starts empty, and each step pushes at

most one symbol), so the maximum length the TM has to travel is $2N$ in each direction, or $4N$ squares total (of course, this is not actually possible, since the stack starts empty and the TM head moves across the input). Hence, there are N steps to simulate, each of which can be simulated in $4N$ steps or less, so the total work of the simulation is in $O(N^2)$. This shows that any language that can be recognized by a DPDA, can be decided by a TM in **TIME(N^2)**.

Problem 4: Hardness Proofs. (30)

a. [10] Prove the language $\left\{0^i 1 0^i | i \geq 0\right\}$ is not regular.

Answer: We prove $A = \left\{0^i 1 0^i | i \geq 0\right\}$ is non-regular by using the pumping lemma for regular languages to obtain a contradiction.

Assume A is regular. Then, according to the pumping lemma, there is some pumping length p such that for any string $s \in A$ where $|s| \geq p$, we can divide $s = xyz$ such that for each $i \geq 0, xy^i z \in A$ and $|y| > 0$ and $|xy| \leq p$.

Choose $s = 0^p 1 0^p$. Since $|xy| \leq p$ all possible divisions of s have xy within the first string of 0^p. Since $|y| > 0$, y must contain at least on 0 and no 1. Thus, pumping y increases the number of 0s before the 1, but does not change the number of 0s after the 1. To be in the language the number of 0s on the left side must equal the number of 0s on the right side, so this produces a string that is not in A. By contradiction, we have shown that A is not regular.

b. [10] Is the language $MORE_{TM}$ defined below undecidable? (Answer clearly, and provide a convincing proof supporting your answer.)

> $MORE_{TM} = \{\langle A, B\rangle\}$ where A and B are descriptions of Turing machines, and the size of the language accepted by A is larger than the size of the language accepted by B.

Answer: Undecidable. Prove by reducing some known undecidable problem to $MORE_{TM}$. We reduce $E_{TM} = \{\langle M\rangle \,| M \text{ is a TM and } L(M) = \oslash\}$, which is known to be undecidable, to $MORE_{TM}$.

If we had a decived for $MORE_{TM}$, we can build a decider for E_{TM} since M is in E_{TM} only if it accepts no strings. Hence, if it accepts more than zero strings, it is not in E_{TM}.

$E_{TM}(\langle M\rangle) =$

> Simulate $MORE_{TM}$ on $\langle M, REJECT\rangle$ where $REJECT$ is a TM that rejects all inputs.
>
> If $MORE_{TM}$ accepts, **reject**. If $MORE_{TM}$ rejects, **accept**.

c. [10] Is the language *NO-SUBSET* defined below in the class NP-Hard? (Answer clearly, and provide a convincing proof supporting your answer.)

> *NO-SUBSET* = $\{\langle\{x_1, x_2, \ldots, x_n\}, m\rangle\}$ where each x_i is a number represented in binary, and there is no subset of the x_i's which sums to m, a number represented in binary.

Answer: Yes. If we had a polynomial time solver for *NO-SUBSET*, we could build a polynomial time solver for *SUBSET-SUM*, which is known to be NP-Complete.

SUBSET-SUM$(\langle\{x_1, x_2, \ldots, x_n\}, m\rangle)$ =

> Simulate *NO-SUBSET* on $\langle\{x_1, x_2, \ldots, x_n\}, m\rangle$. (Given the assumption, this takes polynomial time.)
> If it accepts, **reject**. If it rejects, **accept**.

Thus, we have a polynomial time reduction from *SUBSET-SUM* to *NO-SUBSET*, proving that *NO-SUBSET* is NP-Hard.

Problem 5: Closure. (20)

a. [5] Are the decidable languages closed under concatenation? That is, if A and B are decidable languages, is the language $C = \{ab | a \in A \text{ and } b \in B\}$ a decidable language? (Provide a convincing proof to support your answer.)

Answer: Yes. Since A and B are decidable, there exist TMs M_A and M_B that decide A and B respectively. We can construct a TM M_C that decides C but trying all possible ways of splitting the input w into a and b. (Note that we cannot just simulate M_A first until it accepts, since this can alter the input tape, we wouldn't know how to start M_B). There are a finite (N) number of possible ways of splitting the input. For each possible split, simulate M_A on a. If it accepts, simulate M_B on b. If it accepts, **accept**. Otherwise, try the next possilbe split. If none of the possible splits accept, **reject**.

b. [5] Describe what one would need to do to prove a language A is **not** in the class NP-Complete.

Answer: Either (1) prove A is not in NP, by showing why it cannot be decided by a NDTM in polynomial time, or (2) prove P $\neq$ NP, then prove A is in P.

In *The Limits of Quantum Computers* (your Spring Break reading), Scott Aaronson writes: "If we really could build a magic computer capable of solving an NP-complete problem in a snap, the world would be a very different place: we could ask our magic computer to look for whatever patterns might exist in stock-market data or in recordings of the weather or brain activity. Unlike with todays computers, finding these patterns would be completely routine and require no detailed understanding of the subject of the problem. The magic computer could also automate mathematical creativity. Given any holy grail of mathematics such as Goldbachs conjecture or the Riemann hypothesis, both of which have resisted

resolution for well over a century – we could simply ask our computer to search through all possible proofs and disproofs containing up to, say, a billion symbols."

c. [5] Explain why the proof-finding problem is in NP.

Answer: A proof is a polynomial-time verifiable certificate.

d. [5] Do you agree with the claim that a computer that could solve NP-complete problems in polynomial time could also automate mathematical creativity? (Write a brief paragraph arguing for or against Aaronson's claim.)

Answer: I believe there are stronger arguments *against* the claim (although good arguments supporting it still received full credit). Although a magic NP=P computer could explore an exponential number of possibilities in polynomial time, it cannot explore an infinite number of possibilities in any amount of time. So, it can produce results like, "There is no proof that is shorter than a billion symbols.", but not results like, "There is no proof.". The first type of result leaves the possibility that there is a proof with a billion and one symbols. It does not even mean there is not a short, elegant proof that would be convincing to a mathematician. To explore the proof space, the TM encodes a set of axioms and rules for making inferences from those axioms, and tries all possible ways of using those inference rules starting from the axioms. Many mathematical breakthroughs, however, depend on new proof techniques, not just applying the known inference rules. In theory, all correct proof techniques could be broken down into more primitive inference rules, but the power of the proof techniques is that it enables a short proof. For example, when we use proof-by-induction, all sorts of logical properties about sets and reasoning are embedded in the proof technique, leading to a short an convincing proof. I suspect mathemeticians would find non-proofs generated by such a machine, as enlightening and convincing as arguing that there is no odd perfect number becuase a computer has tested all numbers up to 10^{300} without finding one. Now, if we have a magic computer that could solve undecidable problems, that would really be something!

Problem 6: Busy Bunny. (Bonus)

Is the *BUSY-BUNNY* language defined below NP-Complete? (Prove or disprove.)

> *BUSY-BUNNY* = $\{\langle n, s, k\rangle\}$ where k is the maximum number of 1 symbols that can be on the final tape of a Turing machine with n states and 2 symbols that halts within s steps, starting from a blank input tape.

Answer:

As a starting point, consider the answer to problem 5b: to prove *BUSY-BUNNY* is not in NP-Complete we need to either (1) prove that it is not in NP, or (2) prove that it is in P and P $\neq$ NP. We can prove that it is in NP, so option (1) doesn't work.

There are a finite number of TMs with n states and 2 symbols. We can enumerate them all, and simulate each one for up to s steps. After finishing the simulation, count the number of 1 symbols on the simulated tape. If there is at least one TM that produces k symbols, and

none that produce more than k symbols, the string $\langle n, s, k \rangle$ is in *BUSY-BUNNY*; otherwise it is not. We can simulate a TM running for s steps in polynomial time in s. The number of possible TMs with n states, though, is exponential in n. The problem does appear to be in NP, since we could nondeterministically simulate all of these possible TMs for s steps in polynomial time. However, we cannot simulate all TMs (in a straightforward way) for s steps in polynomial time on a deterministic TM, so it the brute force algorithm for this is not in P. This leaves us believing that perhaps the problem is NP-Complete. To prove it, though, we would need to find a reduction from some known NP-Complete problem to the *BUSY-BUNNY* problem. Alas, no one came up with one that seemed to be convincing, so I still don't know whether or not *BUSY-BUNNY* is NP-Complete, but this is something for you to think about over the summer.

The best answer we got to this question was this one, by Jalysa Conway: