University of Virginia - cs3102: Theory of Computation Spring 2010

Final Exam - Comments

Problem 1: Definitions. (Average 7.2 / 10) For each term, provide a clear, concise, and precise definition from the perspective of a computer scientist.

a. (2) *language*

Answer: A set of strings.

b. (2) *algorithm*

Answer: A procedure that is guaranteed to complete. (A procedure is a precise sequence of steps that can be followed without thought.)

c. (3) **P** (the complexity class)

Answer: The set of languages that can be decided by a deterministic Turing Machine in a number of steps that is polynomial in the size of the input.

d. (3) **NP-Complete**

Answer: The set of languages that are in NP (they have polynomial time verifiers) and NP-Hard (every problem in NP is polynomial-time reducible to every problem in NP-Complete).

Problem 2: Short Answers. (Average 6.6 / 10) Classify each of the following statements as:

True (the statement is correct),
False (the statement is wrong),
Unknown (it is not known to anyone if the statement is True or False)

If the statement is **False** or **Unknown**, your answer should include a brief explanation why. If the statement is **True**, you do not need to include any explanation.

a. (2) If A is a language in **NP**, A is Turing-decidable.

Answer: True

b. (2) If D is a regular language, $D \in \mathbf{P}$.

Answer: True (we can always simulate a DFA in $O(n)$ steps)

c. (2) The language

$$LISKOV = \{w | w \text{ is a string Barbara Liskov said during her talk at UVa }\}$$

is context-free.

Answer: True (the language is finite, so it must be context-free and regular also)

d. (2) There exists some deterministic TM that can decide $3SAT$ in polynomial time.

Answer: Unknown. This depends on whether **P** = **NP**, which is unknown.

e. (2) The language $ADD = \{1^x + 1^y = 1^z | x + y = z\}$ is in **NP-Hard**.

Answer: Unknown. If **P** = **NP** then ADD is **NP-Hard**; otherwise ADD is not **NP-Hard** since we know it is in **P**.

Problem 3: Language Classes. (Average 7.3 / 10)

a. (5) Is the language $\{a^n b a^n | n \geq 0\}$ regular? (A full credit answer *must include a clear and convincing proof* supporting your answer.)

Answer: We prove by contradiction using the pumping lemma.

Assume the language $L = \{a^n b a^n | n \geq 0\}$ is regular. Let p be its pumping length. Choose $s = a^p b a^p$. Since $|s| = 2p + 1 \geq p$, by the pumping lemma, there exists strings x, y, z in L such that $s = xyz$, $|xy| \leq p$, $|y| > 0$, and for all $i \geq 0$ $xy^i z$ is in L. Since $|xy| \leq p$ then both x and y contain only a's. Since $|y| > 0$ then xy^2z has more a's to the left of b than to its right. Thus xy^2z is not in the language. We reached a contradiction. Thus, the assumption that the language is regular is false and we conclude that L is not regular.

b. (5) Is the language $\{a^n b a^n | n \geq 0\}$ context-free? (A full credit answer *must include a clear and convincing proof* supporting your answer.)

Answer: The language $\{a^n b a^n | n \geq 0\}$ is context-free. To prove a language is context-free, we can show a NDPDA or CFG that generates the language. The easiest way is a CFG: $S \rightarrow aSa|b$ generates the language.

Problem 4: Decidability. (Average 3.7 / 10) Prove the language *RECIPROCAL* (defined below) is undecidable.

$$RECIPROCAL = \{\langle M, w\rangle | M \text{ is a description of a TM and } w \in \Sigma^* \text{ and the result of running } M \text{ on } w \text{ is the same as the result of running } M \text{ on } w^R \text{ (the reverse of the string } w) \}$$

That is, a string $\langle M, w\rangle$ is in *RECIPROCAL* if either (1) M accepts w and M accepts w^R or (2) M rejects w and M rejects w^R or (3) M does not halt on w and M does not halt on w^R. (A full credit answer *must include a clear and convincing proof* supporting your answer.)

Answer: We prove by contradiction using a reduction from A_{TM}.

Assume the language *RECIPROCAL* is decidable. Then, there exists a TM, M_R, that decides *RECIPROCAL*. We show how to use it to construct a TM which decides the language $A_{TM} = \{\langle M, w\rangle | M \text{ accepts } w\}$, which we know is undecidable.

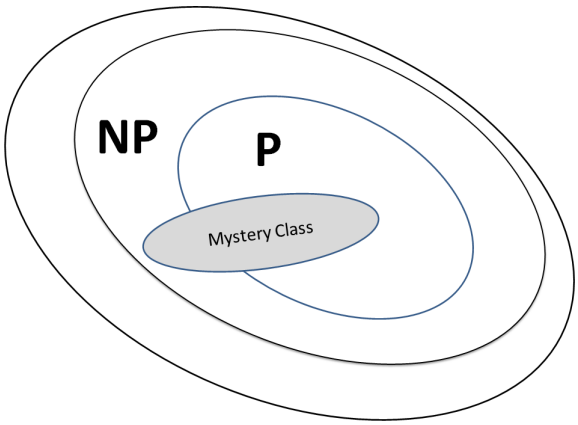
$M_A(\langle M, w\rangle) =$

a. Construct a TM machine M', with input denoted by x that:
   (a) If the input x is w^R, accept.
   (b) Otherwise, simulate M on w and output the result.
b. Run M_R with input $\langle M', w\rangle$ and output the result.

Since M' always accepts w^R then M_R accepts $\langle M', w\rangle$ only when M accepts w. Therefore M_A is a TM that decides A_{TM} which we know is impossible. Therefore, the assumption that *RECIPROCAL* is decidable is false and *RECIPROCAL* is undecidable.

Problem 5: Mystery Class. (Average: 4.6 / 10) For this question, you should assume that "Proof by Cake Deconstruction" has been accepted as a valid proof technique and used to prove **P** $\neq$ **NP**.

Define a complexity class that corresponds to the "Mystery Class" depicted below:



Your class should (1) include infinitely many languages that are in **NP** but not in **P**, (2) include infinitely many languages that are in **P**, (3) exclude infinitely many languages that are in **P**, and (4) exclude infinitely many languages that are in **NP** but not in **P**, and (5) exclude all languages that are not in **NP**. A full credit answer will define the class precisely in terms of a type of machine and its properties.

Answer: Languages that can be recognized by a nondeterministic Turing Machine in polynomial time that do not include ϵ. The machine is a nondeterministic TM with the restriction that all transitions from q_0 on $\sqcup$ must be to the reject state, and there cannot be any ϵ-transitions from q_0. The restriction to a polynomial number of steps is important, otherwise there are language not in **NP** in the class.

Problem 6: Machine Models. (15)

a. (3.8 / 5) A *Right-Only Turing Machine* which is identical to a standard Turing Machine except the tape head can only move right. Describe precisely the class of languages that can be recognized by a *Right-Only Turing Machine*.

Answer: The regular languages.

b. (3.6 / 5) A *Right-Only Cyclic Turing Machine* is like a *Right-Only Turing Machine* except instead of having a one-way infinite tape, the tape is arranged in a cycle. So, when the TM moves right from the last input square, it ends up on the leftmost input square. Prove that a *Right-Only Cyclic Turing Machine* can recognize some languages that are not context free but is *l*ess powerful than a standard Turing Machine.

Answer: (1) A ROC-TM can recognize $a^n b^n c^n$ which is not context free. It can do this by crossing off one a, one b, and one c each time it cycles, and then check all symbols are marked off. One subltety is that the ROC-TM can't tell when it gets to the end of the input, so might not know the difference between *abc* (in the language) and *abcabc* (not in the language). There is an easy solution, though — the ROC-TM knows it starts at the beginning of the input, so add a mark to that square. Then, each time that mark is reached, it knows it has returned to the start.

(2) A ROC-TM cannot recognize $HALTS_{TM}$, which can be recognized by a regular TM. The reason is simulating a TM can require much more tape than the size of its description. For example, consider simulating the TM that is the best currently known BB(6,2) candidate. Its description is short (only six states and the transition function), but its execution writes 3.18710^{10566} symbols on the tape!

c. (2.0 / 5) Is the language $HALTS_{RCTM}$ defined below decidable or undecidable? Include a convincing argument supporting your answer.

$$HALTS_{RCTM} = \{\langle M, w\rangle | M \text{ is a description of a Right-Only Cyclic Turing Machine and } w \in \Sigma^* \text{ and } M \text{ halts on input } w\ \}$$

Answer: Decidable. The number of possible configurations of a ROC-TM is finite: $|Q| \times |\Gamma|^n$. We can simulate an ROC-TM, keeping track of each configuration it enters. After $|Q| \times |\Gamma|^n + 1$ steps, it must have repeated a configuration and will never halt.

d. (bonus +5, average +0.8) Is a Right-Only Cyclic Turing Machine *more powerful* than a NDPDA? (A convincing argument is required for credit.)

Answer: From part b, we know that a ROC-TM can recognize some language that cannot be recognized by an NDPDA. To prove it is more powerful, we also need to show it can recognize every language that can be recognized by an NDPDA. This is non-obvious, since the NDPDA has a stack of infinite depth, thus seems to have more memory than a ROC-TM. The ROC-TM is equivalent to a Linear Bounded Automaton. The restriction of only moving right doesn't really matter, since we can simulate moving left by cycling around to reach the square to the left. A LBA can recognized all context-free languages (in fact, the class of languages recognized by a LBA is the context-sensitive languages, which includes the context-free language).

Problem 7: NP-Completeness.

a. (5.1 / 10) Prove the *ONEORTWO-SUM* language defined below is in NP-Complete.

$$ONEORTWO\text{-}SUM = \{\langle x_1, x_2, \ldots, x_n, k\rangle \mid \text{there exists a set of coefficients, } c_1, \ldots, c_n \text{ where each } c_i \in \{1,2\} \text{ and } \textstyle\sum_{1\le i\le n} c_i x_i = k\}$$

Answer: First, prove *ONEORTWO-SUM* is in **NP**. The list of coefficients is a certificate that can be verified in polynomial time.

Then, to prove it is in NP-Complete, we show *ONEORTWO-SUM* is **NP-Hard** by reducing some known NP-Hard problem to *ONEORTWO-SUM*. The most obvious choices are *SUBSET-SUM* and *KNAPSACK*. If we choose *SUBSET-SUM*, we need to alter the input to *SUBSET-SUM* to allow each element to be used once or twice, instead of zero or one times. This is fairly tricky, and hard to do in a way that gives a lot of confidence that it is correct.

An easier approach, is to modify the reduction from $3SAT$ to *SUBSET-SUM* to require each variable to be used once or twice. So, instead of using $t = 1^l 3^k$, we would use $t = 1^k 6^k$, and change the entries in the bottom half of the table accordingly.

One of the answers to the question of how P=NP will be resolved in Gasarch's survey (http://www.cs.umd.edu/~gasarch/papers/poll.pdf) is:

> **61. Anonymous3:** (Names, schools, dates changed to protect the innocent) On Dec 14, 1991 it was shown that P = NP by undergraduate Mary Lou Koslowsky on her Algorithms final exam at The University of Southern North Dakota. Her ingenious but somewhat hastily written proof, establishing that *3-SAT* could be reduced to *2-SAT* in $O(n^3)$ time, received only 2 points of credit out of a possible 25 and the comment "Wrong." She left computer science and became a pharmacist, working now at Osco Drugs in Lake Wobogon, where all problems have above average complexity.

For this question, you may assume that *2-SAT* is in **P** (which is in fact known to be true).

b. (4.5 / 5) If Ms. Koslowsky's proof was indeed correct, what would it mean to show that *3-SAT* can be reduced to *2-SAT* in $O(n^3)$ time?

Answer: It would prove **P** = **NP**!

c. (3.3 / 5) Suppose instead that Ms. Koslowsky proved that *2-SAT* can be reduced to *3-SAT* in $O(n^3)$ time. What would this mean?

Answer: Nothing whatsoever. It is trivial to reduce *2-SAT* to *3-SAT* in $O(n)$ time: just add a new variable to each clause and two extra clauses to force it to always be false (like we did to reduce 3-SAT to 4-SAT in the last class). But, reducing an easy problem to a hard one is never a worthwhile thing to do (unless it is requested in a silly test question such as 8b).

Problem 8: Double Jeopardy. (15)

a. (3.2 / 5) Prove $n!$ is not in $O(2^n)$.

Answer: Recall the definition of O: $n!$ is in $O(2^n)$ if there exist positive integers c and n_0 such that for every $n \geq n_0$, $n! \leq c2^n$. We prove that for whatever value is choose for c, we can choose an n where $n! > c2^n$. Choose $n = \min(c, 4)$. Since $n! = 1 \times 2 \times 3 \times 4 \times \cdots \times n$ (n terms, all but two of which exceed 2), and $2^n = 2 \times 2 \times \cdots \times 2$ (n terms, all 2), for any choice of c, $c! = c \times c-1 \times \cdots \times 2 \times 1 > c \times 2 \times 2 \times \cdots \times 2$.

b. (5.5 / 10) Prove $3DSAT$ is in **NP-Hard**.

$3DSAT = \{\phi | \phi$ is a satisfiable formula in 3-dcnf $\}$ where 3-dcnf is like 3-cnf, except clauses can be connected using either $\vee$ (or) or $\wedge$ (and).

3-dcnf = strings of the form:
$((v_{0,0} \vee v_{0,1} \vee v_{0,2}) \wedge \ldots \wedge (v_{i_k,0} \vee v_{i_k,1} \vee v_{i_k,2})) \vee (((v_{i_{k+1},0} \vee v_{i_{k+1},1} \vee v_{i_{k+1},2}) \wedge \ldots \wedge (v_{i_m,0} \vee v_{i_m,1} \vee v_{i_m,2}))) \vee \cdots \vee (((v_{i_r,0} \vee v_{i_r,1} \vee v_{i_r,2}) \wedge \ldots \wedge (v_{i_n,0} \vee v_{i_n,1} \vee v_{i_n,2})))$
where $v_{i,j} \in \{x_i | i \geq 0\} \cup \{\overline{x_i} | i \geq 0\}$

Answer: We reduce $3SAT$ to $3DSAT$. If M_{3DSAT} decides $3DSAT$ in polynomial time, we can construct M_{3SAT} to decide $3SAT$ in polynomial time:

(a) Check if the input is a 3-cnf formula. If not, reject.

(b) Simulate M_{3DSAT} on the input and output the result.

This works since 3-dcnf includes all strings in 3-cnf.