COMP 272: Theory of Computation
Spring 2011 Final Exam

1. Print your name and student ID at the top of every page (in case the staple falls out!).
2. This is an open-book, open-notes, open-brain exam.
3. Time limit: 180 minutes.
4. You should answer all the questions on the exam. At least you should read all the questions. Don't be intimidated by the length of the problem — it's not proportional to its difficulty :)
5. When asked to describe a Turing machine, you can use either pseudocode or plain language, just like how you would describe an algorithm. You may assume the most convenient variant of the TM, unless you are explicitly told otherwise.
6. Problems marked with $\star$ may be difficult. Manage your time wisely.
7. You can write on the back of the paper if you run out of space. Please let us know if you need more scratch paper.
8. Now, take a deep breath.

1. (14 pts) Assuming $\mathsf{P} \neq \mathsf{NP}$ and $\mathsf{NP} \neq \mathsf{coNP}$, place the following languages into the correct places in the diagram below.

   (a) $H = \{\text{“}M\text{”}\,\text{“}w\text{”} : \text{Turing machine } M \text{ halts on input string } w\}$.

   (b) $\overline{H}$.

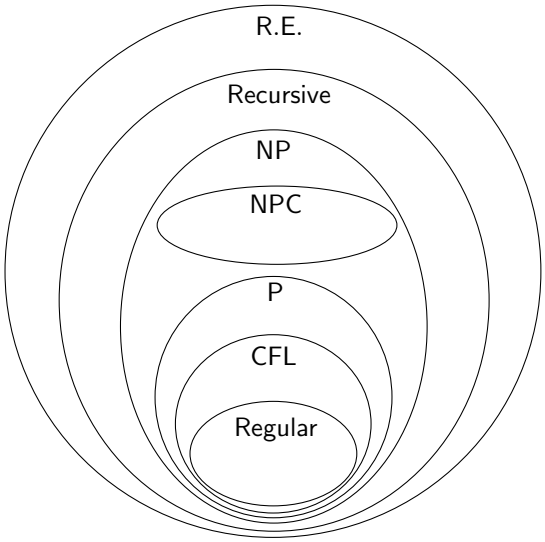
   (c) $\{a\}$

   (d) $\{a^n b^n : n \geq 0\}$.

   (e) $\{a^n b^n c^n : n \geq 0\}$.

   (f) $\{\phi : \phi \text{ is a Boolean formula that is satisfiable}\}$.

   (g) $\{\phi : \phi \text{ is a Boolean formula that is not satisfiable}\}$.



2. (14 pts) For any class of languages C, define $\mathsf{coC} = \{L \mid \overline{L} \in \mathsf{C}\}$. Mark “true”, “false”, or “unknown” for each of the statements below. You do not need to justify your answers.

   (a) $\mathsf{Regular} = \mathsf{coRegular}$.

   (b) $\mathsf{CFL} = \mathsf{coCFL}$.

   (c) $\mathsf{P} = \mathsf{coP}$.

   (d) $\mathsf{NP} = \mathsf{coNP}$.

   (e) $\mathsf{NPC} = \mathsf{coNPC}$.

   (f) $\mathsf{Recursive} = \mathsf{coRecursive}$.

   (g) $\mathsf{R.E.} = \mathsf{coR.E.}$

3. (12 pts) Design a standard (i.e., deterministic, one-tape) Turing machine to perform the following task: Initially on the tape is $\triangleright \underline{\sqcup} \overbrace{1 \ldots 1}^{n} \sqcup \overbrace{1 \ldots 1}^{m} \sqcup$ where $n \geq m$. Compute $n - m$ such that finally on the tape is $\triangleright \underline{\sqcup} \overbrace{1 \ldots 1}^{n-m} \sqcup$.

   (a) (6 pts) Describe informally how the Turing machine works.

   (b) (6 pts) Give a formal definition of the Turing machine. You may either use the graphical notation of Turing machines or give the definition in the form of $M = (K, \Sigma, \delta, s, H)$.

4. (10 pts) Adding more tapes to a Turing machine does not increase its computing power, but it turns out that adding more stacks to a pushdown automaton does. In fact, as long as there are two stacks, a pushdown automaton can be used to simulate a Turing machine. Please fill the missing pieces of the proof below.

   Proof: Define a *deterministic pushdown automaton with two stacks* as $M = (K, \Sigma, \delta, s, H)$, where K is a finite set of states, Σ is an alphabet, $s \in K$ is the initial state, $H \subseteq K$ is the set of halting states, and $\delta : K \times \Sigma^* \times \Sigma^* \to K \times \Sigma^* \times \Sigma^*$ is the transition function. The *configuration* of M is a triple $(p, \alpha, \beta) \in K \times \Sigma^* \times \Sigma^*$, where p is the current state, and α and β are the contents of the two stacks, read top-down. For convenience, we assume that the input is given in the second stack (there is no input tape), and Σ includes $\{\triangleright, \sqcup, \triangleleft\}$, where $\triangleright$ and $\triangleleft$ are the "bottom markers" of the two stacks. More precisely, on an input string $w \in (\Sigma - \{\triangleright, \sqcup, \triangleleft\})^*$, the initial configuration is $(s, \sqcup\triangleright, w\triangleleft)$. When the current configuration is $(p, \gamma\alpha, \eta\beta)$ for $p \in K - H$, $\alpha, \beta, \gamma, \eta \in \Sigma^*$, M will move to the configuration $(q, \gamma'\alpha, \eta'\beta)$, if $\delta(p, \gamma, \eta) = (q, \gamma', \eta')$; otherwise the automaton halts.

   Now, given a Turing machine $M_1 = (K, \Sigma, \delta_1, s, H)$, whose initial configuration is $(s, \triangleright\underline{\sqcup}w)$ on input string w, we will build a deterministic pushdown automaton with two stacks $M_2 = (K, \Sigma, \delta_2, s, H)$ that will faithfully simulate the actions of M_1, that is, M_1 reaches a state q when and only when M_2 reaches q. The two machines share the same M, Σ, s, and H. The transition function δ_2 for M_2 is defined as follows.

   For all $q \in K - H, \sigma \in \Sigma$, if $\delta_1(q, \sigma) = (p, \sigma')$, then

   [

   ]

   For all $q \in K - H, \sigma \in \Sigma$, if $\delta_1(q, \sigma) = (p, \rightarrow)$, then

   [

   ]

   For all $q \in K - H, \sigma \in \Sigma$, if $\delta_1(q, \sigma) = (p, \leftarrow)$, then

   [

   ]

5. (12 pts) Show that a language L is recursive if and only if there is a Turing machine M that enumerates the strings of L in a non-decreasing order of their lengths.

6. (15 pts) Show that the following problem is undecidable: Given a Turing machine M and a string w, determine whether M ever makes two consecutive left moves when started with input w.

7. (8 pts) Recall that a language $L \in \mathsf{NP}$ if there exists a nondeterministic Turing machine M that decides L in polynomial time, where "M *decides* L" means that

   (a) For any $w \in L$, at least one computation path of M reaches the "yes" state;

   (b) For any $w \notin L$, all computation paths of M reach the "no" state.

   Now let us change the definition of *decide* to the following, which lead to different classes of languages.

   The class RP: We say that M RP-*decides* L if

   (a) For any $w \in L$, at least 2/3 of the computation paths of M reach the "yes" state;

   (b) For any $w \notin L$, all computation paths of M reach the "no" state.

   Correspondingly, RP is the class of languages that can be RP-decided by a nondeterministic Turing machine in polynomial time.

   The class BPP: We say that M BPP-*decides* L if

   (a) For any $w \in L$, at least 2/3 of the computation paths of M reach the "yes" state;

   (b) For any $w \notin L$, at least 2/3 of the computation paths of M reach the "no" state.

   Correspondingly, BPP is the class of languages that can be BPP-decided by a nondeterministic Turing machine in polynomial time.

   The class ZPP: Now in addition to the "yes" and "no" states, there is another possible final state "do not know". We say that M ZPP-*decides* L if

   (a) For any $w \in L$, at least 2/3 of the computation paths of M reach the "yes" state, while all the other computation paths reach the "do not know" state.

   (b) For any $w \notin L$, at least 2/3 of the computation paths of M reach the "no" state, while all the other computation paths reach the "do not know" state.

   Correspondingly, ZPP is the class of languages that can be ZPP-decided by a nondeterministic Turing machine in polynomial time.

   Please put $\mathsf{RP}, \mathsf{BPP}, \mathsf{ZPP}, \mathsf{P}, \mathsf{NP}$ into the following blanks. You do no need to justify your answer.

$$\_\_\_\_\_ \subseteq \_\_\_\_\_ \subseteq \_\_\_\_\_ \begin{cases} \subseteq \_\_\_\_\_ \\ \subseteq \_\_\_\_\_ \end{cases}$$

   FYI: $\mathsf{RP}, \mathsf{BPP}, \mathsf{ZPP}$ are important classes of problems that can be solved by *randomized* algorithms. Also, the fraction 2/3 is not critical; it can be replaced by any constant between 1/2 and 1 (both noninclusive).

8. (15 pts)$\star$ Consider the following solitaire game. You are given an $m \times n$ board where each cell is empty or occupied by either a red stone or a blue stone. Your goal is to remove a subset of the stones so that the following two conditions are met: (a) each column has only red stones or only blue stones or nothing, and (b) each row has at least one stone (but may have stores of different colors). Achieving this objective may or may not be possible depending upon the given configuration. Define SOLITAIRE to be the problem of deciding if this objective is achievable for a given configuration. Prove that SOLITAIRE is NP-complete. (Hint: Reduce from 3SAT.)