COMP 272: Theory of Computation
Spring 2011 Final Exam

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) $\overline{H}$.

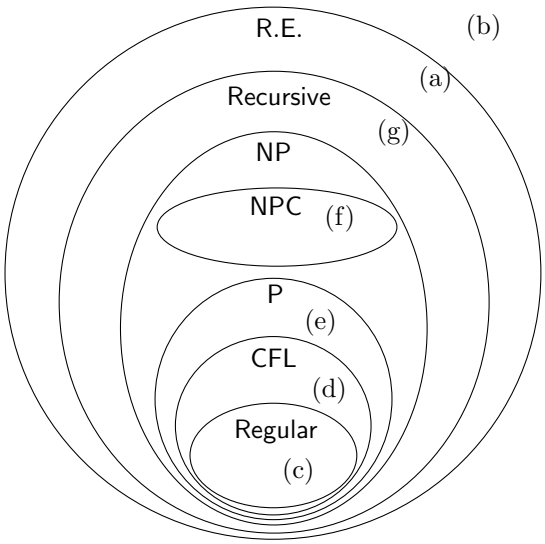
   (c) $\{a\}$

   (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) $\mathsf{Regular} = \mathsf{coRegular}$, **True**;

   (b) $\mathsf{CFL} = \mathsf{coCFL}$, **False**;

   (c) $\mathsf{P} = \mathsf{coP}$, **True**;

   (d) $\mathsf{NP} = \mathsf{coNP}$, **Unknown**;

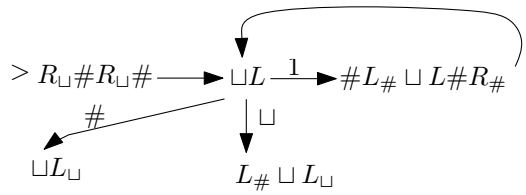
   (e) $\mathsf{NPC} = \mathsf{coNPC}$, **Unknown**;

   (f) $\mathsf{Recursive} = \mathsf{coRecursive}$, **True**;

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

3. (12 pts) Design a one-tape Turing machine to perform the following task: Initially on the tape is $\triangleright\underline{\sqcup}\overbrace{1\dots1}^{n}\sqcup\overbrace{1\dots1}^{m}\sqcup$ where $n \geq m$. Compute $n - m$ such that finally on the tape is $\triangleright\underline{\sqcup}\overbrace{1\dots1}^{n-m}\sqcup$. First describe informally how the Turing machine works, then give a formal definition. You may either use the graphical notation of Turing machines or give the definition in the form of $M = (K, \Sigma, \delta, s, H)$.

Solution: for clarity, we will use special symbol # to imply where we are operating. And our method is following: we first set up two #s following the end of two substrings like: $\triangleright \sqcup 1 \dots 1\#1 \dots 1\underline{\#}$, and let the two #s move left and eat up exactly one 1 in turn and recursively until the latter substring is eat up, then we erase the two # and move the tape header at the right place and halt.Please see the graph below as a formal definition:



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 $(p, a\alpha, b\beta)$ for $p \in K - H$, $\alpha, \beta \in \Sigma^*, a, b \in \Sigma$, M will move to the configuration $(q, \alpha'\alpha, \beta'\beta)$, if $\delta(p, a, b) = (q, \alpha', \beta')$; 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

   [

   $$\delta_2(q, \sigma, e) = (p, \sigma', e)$$

   here, e means empty string.]

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

   [

   $$\delta_2(q, \sigma, \tau) = \begin{cases} (p, \tau\sigma, e) & : \quad \tau \neq \triangleleft \\ (p, \sqcup\sigma, \triangleleft) & : \quad \tau = \triangleleft \end{cases}$$

   ]

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

   [

   $$\delta_2(q, \sigma, e) = (p, e, \sigma)$$

   since by convention $\delta_1(q, \triangleright) = (p, \leftarrow)$ is illegal.]

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.

Solution: " $\Leftarrow$ ", suppose there is a Turing machine M that enumerates the strings of L in a non-decreasing order of their lengths. Then we can construct a Turing machine M' that can decide L. The Turing machine M' performs like: for any input w, it simulates M until M is just enumerating w or has enumerated all the strings of L with length no bigger than $|w|$. If M is just enumerating w, M' accept w; if M has finished enumerated all the strings with length no bigger than $|w|$, M' rejects. **Verify**: for any $w \in L$, since M will enumerate w sometime, then M' will accept it. For $w \notin L$, when M has enumerated all strings with length no bigger than $|w|$, M' will reject correctly. Thus M' can decide L. In another word, L is recursive.

" $\Rightarrow$ ", suppose there is a Turing machine M' than can decide L. We come to construct a Turing machine M like this: by using a non-decreasing order of lengths of strings and within the same length, using lexicographical order, we iterate each string, and for each string w, we run M' on w; when M' accepts it, M enumerates it; otherwise skips it and continues. **Verify**: for any string $w \in L$, it will eventually be iterated. Since it can be accepted by M', it will be enumerated by M. And the order of enumeration is an obviously non-decreasing order of lengths. And for any string $w \notin L$, since it will never be accepted by M', M will not enumerate it. So such M exists.

It got proved.

6 We reduce the HALTING problem to this one. For any input "M" "w", we construct a TM M' which does the following: M' simulates the behavior of M on any input, except that: (1) it adds a dummy step (say writing the symbol it reads to the current position) whenever M makes a left move; and (2) when M halts, M_w makes two consecutive left moves and then halts. It is easy to verify that M halts on w iff M_w makes two consecutive left moves on the same input w. So the problem is undecidable.

7 P, ZPP, RP, BPP, NP

8 The certificate for this problem is the stones that are removed, and we need to check whether the final configuration meets the requirement. This clearly can be done in polynomial time.

To prove its NP-completeness, we will reduce $3SAT$ to this problem. Given any $3CNF$, we construct the initial configuration for solitaire game as follows. Each row represents a clause, and each column represents a variable. We put a blue stone in the position B_{ij} if the ith clause contains the jth variable, and put a red stone in B_{ij} if the ith clause contains the complement of the jth variable. Next we show the $3CNF$ is satisfiable iff the configuration we constructed has a solution.

For the only if part, we can construct the solution for solitaire game from the satisfying assignment as follows. If a variable in the assignment is TRUE then remove all the red stone in the corresponding column, otherwise remove all the blue stone in the column. It is easy to check that if the value for a variable or its complement is TRUE in a clause, the stone in the corresponding position left. Since each clause has at least one variable or its complement satisfied, the corresponding row has at least one stone left.

For the if part, we can construct the satisfying assignment from the final configuration similarly. If a row only contains blue stones, we assign TRUE to the corresponding variable, and vice versa. It can be easily verified that the assignment satisfies the $3CNF$ formula.