Name________________________

CS 383
Final Exam Solutions
December 13, 2017

The exam has 8 questions. #7 is worth 16 points; the other seven are worth 12 points each

1. Identify the seven languages below as
   - R = regular
   - C = context-free but not regular
   - D = recursive (decidable) but not context-free
   - E = recursively enumerable but not recursive
   - N = not recursively enumerable

   You don't need to justify your answers.

   a. $\{ (01)^n \mid n >= 0\}$ For example, 010101 is in this language
   Regular

   b. $\{(0^n1)^n \mid n>0\}$ For example, 000100010001 is in this language.
   Recursive

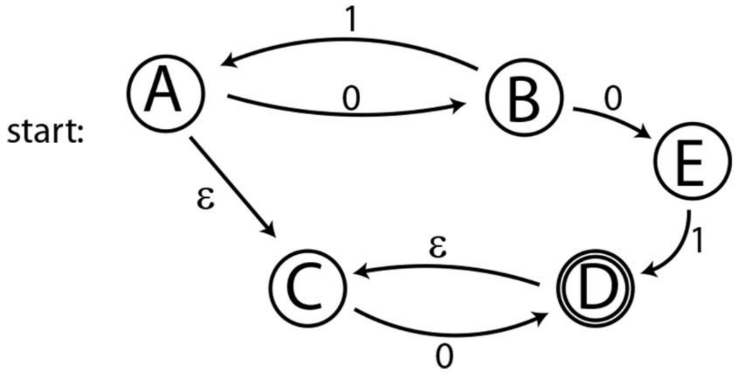
   c. Strings of the digits 0 thru 9 where no digit appears more than two times.
   Regular

   d. Strings of the form *ww*, where *w* is a string of 0s and 1s.
   Recursive

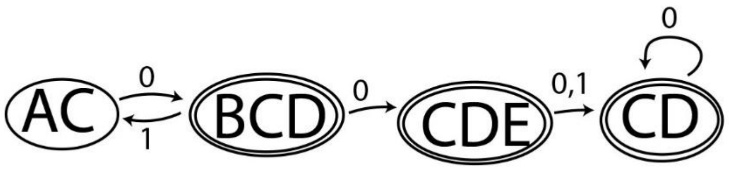
   e. {m | m is a valid encoding of a Turing Machine} (Remember that we encoded a transition $\delta(q_i,t_j)=(q_k,t_L,d_m)$ as $0^i10^j10^k10^L10^m1$ and encoded the TM as a sequence of transitions followed by the final state).
   Regular

   f. The set of encodings of Turing Machines that *do* accept their own encodings. Don't confuse this with the diagonal language, which is the set of TMs that *don't* accept their own encodings.
   Recursively enumerable

2. Here is an ε-NFA with A as its start state (the label didn't position quite right).



a) Convert this to a DFA.



b) Describe in English the strings that are accepted by these automata,

This accepts (01)*00* + (01)*0010* These strings have prefix (01)*, followed either by 0 or by 001, and then with a suffix consisting of any number of 0s. That is more or less English.

3. Is the language of strings of 0s and 1s that have different numbers of 0s than 1s regular? For example 001, 0101010 and 111 are all in this language. Either prove the language is regular or prove that it isn't.

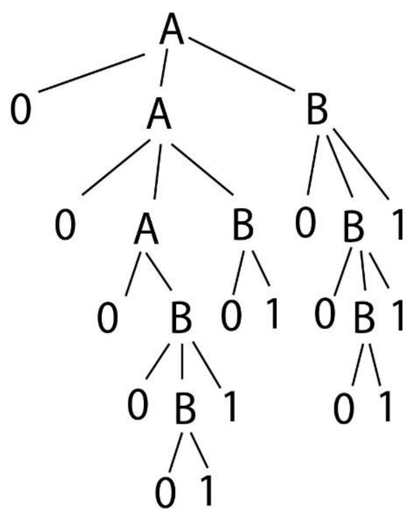
   No, this language is not regular. If it was, its complement, which is the language of strings of 0s and 1s with the same number of 0s and 1s, would also be regular, and we showed the latter isn't.

4. Consider the language $\{0^n(0^m1^m)^n \mid n > 0, m>0\}$ Just to be clear, strings in this language start with n 0s. They then have n groups, where each group consists of some number of 0s followed by the same number of 1s. For example, 000001101000111 is in this language.

   a) Give a grammar for this language.

      A => 0AB| 0B
      B => 0B1 | 01

   b) Give a parse tree for the derivation of 000001101000111 with your grammar.



5. Show that the language $\{0^n 1^m 2^n \mid n > 0, m < n\}$ is not context-free.

Suppose the language is context-free. Let p be its pumping constant. Consider the string $z = 0^{p+1}1^p2^{p+1}$. This is longer than p so it should be pumpable. Let z=uvwxy be any decomposition with vx not empty and $|vwx| < p$. Since there are p 1s, v and x cannot have both 0s and 2s. If v and x contain 0s but not 2s, pumping leaves us with different numbers of 0s than 2s. The same applies of v and x contain 2s but not 0s. The only other possibility is that vwx consists entirely of 1s. But then uv^2wx^2y no longer has more 0s and 2s than 1s. Any way we slice it, string z cannot be pumped. This contradicts the assumption that the language is context-free.

6. Describe a TM that takes as input a string of n 0s and halts with 2^n 0s on its tape. You can use as many tapes as you want, though the number of tapes needs to be a constant and can't depend on n. It is not necessary to give all of the machine's transitions; just break this down to simple steps that can clearly be performed by a Turing Machine.

   Here is a 3 tape TM that does it. Start with the input 0n on tape 1 and a single 0 on tape 2 and a single 0 on tape 3. Erase a 0 from tape 1, then copy all of tape 2 to the end of tape 3. (Overwrite a 0 on tape 2 with X, write a 0 on tape 3; continue until there are no 0s on tape 2.) Now erase everything on tape 2, then copy all of tape 3 onto tape 2. Now erase another 0 from tape 1, copy tape 2 to the end of tape 3, erase tape 2, and copy tape 3 to tape 2. Continue this until tape 1 is empty, then copy tape 3 to tape 1 as the answer.

7. Let $\mathcal{L}_{\text{hippy-dippy}}$ be the set of encodings of Turing Machines that accept all strings. Our friend Happy (actually, his encoding) is a member of $\mathcal{L}_{\text{hippy-dippy}}$. The complement of $\mathcal{L}_{\text{hippy-dippy}}$ is $\mathcal{L}_{\text{skeptical}}$, the set of Turing Machines that fail to accept at least one string. Rice's Theorem tells us that neither of these sets is Recursive. Are either of them Recursively Enumerable? You can use facts we proved in class about the Diagonal language, the Universal language, the Halting language, and the Empty and Non-Empty languages (and the complements of any of these). Anything else you use you need to prove.

a) Either prove that $\mathcal{L}_{\text{hippy-dippy}}$ is Recursively Enumerable or prove it isn't.

This is not Recursively Enumerable. We can reduce the complement of the halting language to $\mathcal{L}_{\text{hippy-dippy}}$. to see this, start with an (M, w) pair. Create a new Turing Machine M' that runs on input x as follows: M' simulates M on w for |x| steps. If M halts on w within |x| steps, M' rejects x; otherwise M' accepts x. M does not halt on w exactly when M' is in $\mathcal{L}_{\text{hippy-dippy}}$. So a TM that recognizes the latter would recognize the complement of the halting language, and we know the latter is not recursively enumerable.

b) Either prove that $\mathcal{L}_{\text{skeptical}}$ is Recursively Enumerable or prove it isn't.

This one is easier. We reduce the complement of the universal language to this. Given an (M, w) pair create a new Turing Machine M'. For any input s, M' accepts s if s is not w, and M' simulates M on w if s is w. M' accepts all strings if an only if M accepts w. So if we could recognize if M' is in $\mathcal{L}_{\text{skeptical}}$ then we can recognize that (M, w) is in the complement of the universal language, which we know we can't do.

8. Explain in English what Cook's Theorem (aka the Cook-Levin Theorem) means, without using the terms $\mathscr{P}$, $\mathscr{NP}$, NP-Complete and NP-Hard

   Cook's Theory says that if we could decide in polynomial time whether an arbitrary Boolean expression is satisfiable ("polynomial time" means the number of steps in the decision process is bounded by a polynomial function of the length of the Boolean expression), then we could decide in polynomial time anything that can be verified (non-determinstically decided) in polynomial time.