Hints for Exam Review Questions

If you are really stuck on one of the review questions, then you can refer to these hints for that question. But doing so REALLY reduces the value (in terms of preparing for the exam) of working on the problem and solving it yourself. You won't be getting hints during the exam, obviously. So you should only look here after spending a lot of time on the review questions yourself first.

Regular Languages

1. Think about what pieces of information your DFA has to remember as it processes the string so that it can decide at the end whether the entire string is in L_1. You should remember as little information as you can.

2. It might actually be easiest to design a DFA for this language. (Remember that a DFA is also an NFA.)

3. For L_2, it helps to first reformulate the definition of the language to describe what strings *are* in the language (rather than which strings are not in the language).

4. Use the Pumping Lemma.

5. This language is the union of two simpler languages.

6. There are several ways to solve this question. One would be to inductively construct a regular expression for $\textsc{Prefix}(L)$, given a regular expression for L. But the easiest way is to modify the DFA for L to accept $\textsc{Prefix}(L)$.

7. Use induction on the number of steps the machine has taken to prove that after processing an even number of characters, the machine is in a non-accepting state.

8. Recall that we can construct a DFA that accepts the intersection of the languages accepted by two given DFAs.

Context-Free Languages

9. Your CFG should ensure that each 2 added to the end of the string has a matching 0 or 1 added to the beginning of the string. Then there should be some extra 0's and/or 1's.

10. The stack is useful for checking the constraint that $m \geq n$. The constraint that n is even can be checked without using the stack (since this property can be checked by a DFA).

11. First, let's look at what can be generated by T. The following two claims can each be proved by induction on j.

Claim 1: For all $j \geq 0$, T generates string $a^j b^j$.

Claim 2: For all $j \geq 0$, if T generates a string $x \in \{a,b,c\}^*$ in $j+1$ steps, then $x = a^j b^j$.

Now we know that the set of strings of terminals that can be generated from T is exactly $\{a^j c^j : j \in \mathbb{N}\}$.

Stop reading here, and try to finish the proof. Further hints below.

Next, tackle what can be generated from S. Similar proofs by induction on k establish the following claims:

Claim 3: For all $k \geq 0$, $S \overset{*}{\Rightarrow} a^k T c^k$.

Claim 4: For all $i \geq 0$, if S generates a string $x \in \{a,b,c\}^*$ in $i+2$ steps, then there exists a $j \geq 0$ such that $x = a^i b^j c^{i-j}$.

From claims 1 and 3, we see that for all j and k, S can generate $a^k a^j b^j c^k = a^i b^j c^k$ (where $i = j+k$, or $i - j = k$).

From claim 4, we see that all strings generated by the grammar are of the form $a^i b^j c^k$, where $i - j = k$.

12. You could use a PDA that whose stack stores a representation of $i - j - k + \ell$.

To design a CFG (which is probably trickier), notice that the constraint says $i + \ell = j + k$. Break the language into the union of two languages: the subset that has $i \geq j$ and the subset that has $i \leq j$.

13. It is.

14. It isn't.

Decidable and Recognizable Languages

15. It's probably easiest to use two tapes. Copy x to the second tape and try all possible starting locations within y to see if you get a match.

16. To check if $x \in L_1 \cap L_2$, run the decision algorithm for testing whether $x \in L_1$ and then run the decision algorithm for testing whether $x \in L_2$.

17. Think about the algorithm we used in class to recognize $\overline{E_{TM}}$. You can use a similar idea to design an algorithm that recognizes L_{17}.

18. Show that if you could decide this language, you could decide A_{TM}.

19. The answer to this question follows immediately from the answers to the previous two questions. (Take a look at your lecture notes from the last class.)

20. You can use an idea from the proof of Theorem 4.22 to answer this question.

21. It is decidable (and hence recognizable).

22. $\overline{L_{22}}$ is recognizable.

23. To show that L_{23} is recognizable, see the hint for question 19.

24. This is similar to the proof we did in class to prove that the integers are countable.

25.

(a) This question is a bit difficult. We can use an idea similar to the algorithm we studied in class to check whether a string is generated by a grammar. Define a 3-dimensional boolean array M such that $M[i, j, k]$ is 1 if the first i characters of a can be spliced together with the first j characters of b to obtain the first k characters of c, and 0 otherwise. Then describe how to fill in the array.

(b) Build a DFA for $\textsc{Splice}(L_1, L_2)$, given DFAs for L_1 and L_2. The new DFA's state should be an ordered pair of states of the 2 original DFAs.

(c) The simplest decision algorithm is probably to use brute force: given an input string c, try all possible strings a, b (of length at most $|c|$) and see if they can be used to show $c \in \textsc{Splice}(L_1, L_2)$. (This will require using the decision algorithms for L_1 and L_2 and the answer to part (a).)