(1) Give a decision procedure for the following computational problem (12 pts.):

**INSTANCE:** A DFA $M$.

**QUESTION:** Does $L(M)$ equal its reversal?

**Answer:** Pseudocode as in the text like the following is fine:

1. Make an NFA $N$ as in lecture such that $L(N) = L(M)^R$.
2. Convert $N$ into an equivalent DFA $M'$. Then $L(M) = L(M^R) \iff L(M) = L(M')$.
3. Build a DFA $M''$ such that $L(M'') = L(M) \triangle L(M')$ by carrying out the Cartesian Product construction for the symmetric difference operation ($\triangle$ or $\oplus$; the text doesn’t give a symbol) as in Theorem 4.5.
4. Run the decider for $E_{DFA}$ on $\langle M'' \rangle$. If it accepts, accept $\langle M \rangle$, else reject. This is correct since $L(M) = L(M^R) \iff L(M) = L(M') \iff L(M'') = \emptyset \iff \langle M'' \rangle \in E_{DFA}$.

(2) Take the algorithm for determining the set $N$ of variables that derive $\epsilon$ (called NULLABLE in the Chomsky NF notes on the course webpage) and initialize $N = \Sigma$ instead of $N = \emptyset$. Explain in your own words your understanding of how this now gives you the procedure in Theorem 4.8 for deciding $E_{CFG}$. Then apply it to solve the text’s problem 4.28 on page 212 (in the second edition, it is 4.26 on page 184): Sketch in prose a decision procedure for the following problem (9 + 18 = 27 pts.):

**INSTANCE:** A CFG $G = (V, \Sigma, R, S)$, and a string $x \in \Sigma^*$.

**QUESTION:** Does $G$ generate some string $w \in \Sigma^*$, such that $x$ is a substring of $w$?

**Answer:** My own explanation was conveyed in lecture and on Piazza. At the end is a brief little extra to illustrate sound and comprehensive. Now to decide the text’s problem:

1. Form the regular expression $R = \Sigma^*x\Sigma^*$. Then a string $w$ has $x$ as a substring iff $w \in L(R)$.
2. Calculate the CFG $G_R$ such that $L(G_R) = L(G) \cap L(R)$.
3. Run the decider for $E_{CFG}$ on $\langle G_R \rangle$ and accept $\langle G, x \rangle$ iff the decider rejects.

This works because the answer to the original problem is ‘yes’ if and only if $L(G) \cap L(R) \neq \emptyset$, which $\iff L(G_R) \neq \emptyset$.

(3) Consider the following decision problem:

**INSTANCE:** A deterministic one-tape Turing machine $M$ and an input $x$ to $M$.

**QUESTION:** Does $M$ on input $x$ at some point overwrite a non-blank char by the blank?
Explain what lectures have already said about the ability to code one-tape TMs so they only write blanks over non-blank chars during a final “good housekeeping” stage before accepting, and explain why deciding this problem would achieve the impossible by deciding $A_{TM}$. (You need not use a “reduction” per-se. 15 pts., for 54 on the set.)

**Answer:** Again as said in lecture and on Piazza, the idea is to convert an arbitrary TM $M = (Q, \Sigma, \Gamma, \delta, B, s, q_{acc}, q_{rej})$ into an equivalent TM $M'$ by first letting % stand for some new symbol not already in $\Gamma$. (If $\Gamma$ already includes all of UNICODE then % could be the symbol for Prince.) For every instruction $(p, c/d, D, q)$ with $c = B$, add $(p, %/d, D, q)$, and then for every one with $d = B$, replace it by $(p, c/%, D, q)$. This treats % the same as the blank, but avoids actually writing the blank (over a non-blank char or otherwise). Finally make all arcs to $q_{acc}$ go first to two new states that find the right end of the island of non-blank chars (which are all connected thanks to any use of %) and write real blanks over all of them before going to $q_{acc}$.

Thus $M'$ accepts any input $x$ iff $M$ does, but also has the property that it blanks out nonblank chars when and only when it accepts.

Suppose we had a decider $Q$ for the above problem. Then we could build a decision procedure $R$ that given an arbitrary instance $\langle M, x \rangle$ of the $A_{TM}$ problem, converts $M$ to $M'$ as above, runs $Q$ on $\langle M', x \rangle$, and accepts iff $Q$ accepts. Then $R$ is total, and $R$ accepts $\langle M, x \rangle \iff M'$ on input $x$ overwrites a non-blank char by $B$ (in the final routine) $\iff M'$ accepts $x \iff M$ accepts $x \iff \langle M, x \rangle \in A_{TM}$. So $R$ is total and $L(R) = A_{TM}$, which would make $A_{TM}$ decidable. But we showed that $A_{TM}$ is undecidable. This contradiction shows that $Q$ cannot exist, so the above problem is undecidable.

**Footnote to (2):** Here’s a little more than the text says in its proof of Theorem 4.8 on why the algorithm for $E_{CFG}$ is correct: The first pass identifies those rules $A \rightarrow u$ with $u \in \Sigma^*$ and adds to $N$ all such $A$. If then we have another rule $B \rightarrow X$ where $X \in N^*$, it is sound to add $B$ to $N$ because the entire right-hand side can derive a terminal string. The algorithm stops only when an entire pass through the remaining rules fails to add a new variable. Suppose the algorithm failed to be comprehensive. Then there would be some variable $B$ that derives a terminal string $v$ but never got added to $N$. In the (or some) parse tree for $v$, the children of $B$ at the root form the right-hand side $X$ of the first rule used. Either $X \in N^*$ or $X$ includes some variable $B'$ that is not in $N$. In the latter case, go to $B'$ and let $X'$ be its children. Eventually we must reach some $B'' \notin N$ whose children $X''$ are in $N^*$, since that condition holds at the bottom with any part of $v$. But in that case the algorithm wouldn’t have stopped without adding $B''$ to $N$. This contradiction shows that the final $N$ produced by the algorithm is correct. Thus in particular, $L(G) \neq \emptyset \iff S \in N$, so it decides $E_{CFG}$ (and what I’ve tended to call $NE_{CFG}$ for the complementary “nonempty?” problem).

[As a further note for Tuesday’s lecture, since the algorithm either adds at least one new variable at each pass or halts, and each pass goes thru each rule, the running time is $O(|V| \cdot |R|)$, which is polynomial time in the size roughly $|V| + |R|$ of the grammar description.]