(1) Consider the following decision problem: Given a CFG $G = (V, \Sigma, R, S)$ with $\Sigma = \{a, b\}$, is $L(G) \cap a^+ \neq \emptyset$? That is, does $S$ generate one or more strings that consist only of one or more $a$’s? Sketch a decision procedure in prose similar to what the text does in section 4.1 with $E_{CFG}$ and what my lectures did when comparing the latter to the algorithm for whether $\epsilon \in L(G)$.

(You may if you wish use the conversion from $G$ to a $G'$ without $\epsilon$-rules as the first step of your procedure with no other comment needed on how to do it, but further steps should show all details including sketching any while-loops that may be needed. 24 pts.)

Answer: The first main idea is to call a variable “a-able” if it can derive a string in $a^+$. Initialize a set A-ABLE to \{a\} (OK, a is a terminal but we will only be adding variables to it). Then we execute the following loop:

```plaintext
bool changed = true;
while (changed) {
    changed = false;
    for (each rule D --> X with D not in A-ABLE) {
        if (X is in (A-ABLE)^+) {
            A-ABLE += {D};
            changed = true;
        }
    }
}
if (S is in the final A-ABLE) { accept; } else { reject; }
```

There is however a hitch compared to the similar algorithms for the problems of whether $S$ can derive $\epsilon$ or whether $L(G) \neq \emptyset$. This is shown by a rule such as $D \rightarrow AE$ where $A$ is in A-ABLE but $E \rightarrow \epsilon$. In that case, $D$ should be added to A-ABLE but we will never do so because $E$ isn’t. But we don’t want to use A-ABLE* in the while-loop because we don’t want to ultimately match $\epsilon$—we need a string in $a^+$.

The IMHO-cleanest solution is to remove $\epsilon$-rules from the grammar before beginning the loop. Do the step from $G$ to $G'$ as was hinted. Then no variable in the modified $G'$ is nullable and the hitch never happens.

(2) Consider the following decision problem:

**Instance:** A DFA $M = (Q, \Sigma, \delta, s, F)$ and two strings $u, v \in \Sigma^*$.

**Question:** Does $L(M)$ contain all strings that can be formed by concatenating $u$ and $v$ as often as desired in any order?

(a) Give in pseudocode a decision procedure for this problem. (18 pts.)

(b) With reference to the above notes, tweak your procedure so that it solves the somewhat more general problem where $M$ can be given as a DPDA not just a DFA. (6 pts.)
(c) With reference to the above notes and facts in the lecture notes it references, say why the problem becomes undecidable when \( M \) is allowed to be an NPDA not just a DPDA.

(6 pts., for 30 total and 54 total on the set)

Answer: (a) The condition on strings in the Question: is captured by defining \( A = (u + v)^* \). (Or if you don’t consider “zero times” as allowed by the meaning of “often” then you get \( A = (u + v)^+ \), which works equally well for the whole question. In office hours I avoided giving away the regular expression by calling \( A \) “ALLUVIAL”—all \( u, v \)-ial, geddit?) Then the Question: asks, is \( L(M) \supseteq A \)? The answer is yes if and only if \( A \setminus L(M) = \emptyset \). Here is the resulting algorithm, which works “by reduction to \( E_{DFA} \)”: 

Given the DFA \( M \) and strings \( u,v \) in \( \{a,b\}^* \):
1. Form the regular expression \((u + v)^*\).
2. Convert \((u + v)^*\) into a DFA \( M' \).
3. Use the Cartesian Product construction with \( \setminus \) (== X AND NOT Y) as the Boolean operation to create a new DFA \( M'' \).
4. Run the text’s algorithm for \( E_{DFA} \) on \( \langle M'' \rangle \) and accept iff it accepts.

(b) The game-plan and first two steps are as above. For step 3, we need a DPDA \( M'' \) such that \( L(M'') = A \setminus L(M) \). Can we get one? Well, DCFLs are closed under complement so we can get a DPDA \( \hat{M} \) such that \( L(\hat{M}) = \sim L(M) \). And DCFLs are closed under \( \cap \) with a regular set so we can get a DPDA \( M'' \) such that \( L(M'') = L(\hat{M}) \cap A \), which equals \( A \setminus L(M) \) which is what we need. Finally, in place of step 4, we convert \( M'' \) into an equivalent CFG \( G'' \) and run the text’s algorithm for \( E_{CFG} \) on \( \langle G'' \rangle \), accepting \( \langle M, u, v \rangle \) if and only if that algorithm accepts \( \langle G'' \rangle \). Since each step is guaranteed to halt for any input, this is a decision procedure.

(c) With \( M \) now being a possibly nondeterministic PDA, we’d still be in business if we could compute a CFG \( G'' \) such that \( L(G'') = A \setminus L(M) \). The above plan doesn’t work because \( \sim \) \( L(M) \) might not be a CFL. Can the problem be solved another way? The answer is definitely not: Suppose we had a decision procedure \( A \) for our problem (c). Given any NPDA \( M \), let us run \( A \) on \( \langle M, u, v \rangle \) where we fix \( u = a \) and \( v = b \). Then \( A(\langle M, a, b \rangle) \) is deciding whether \( L(M) \supseteq (a + b)^* \), i.e., whether \( L(M) = \Sigma^* \). This would enable to decide the ALLCFG problem on any grammar \( G \) by converting \( G \) into an NPDA \( M \) and running \( A(\langle M, a, b \rangle) \). However, as presented but not proved in lecture, the ALLCFG problem is undecidable. Hence \( A \) cannot exist, which means that (c) too is undecidable.

If you use \((u + v)^+\) then in (c) taking \( u = a \) and \( v = b \) gives you the problem of whether any given NPDA accepts \( \Sigma^* \setminus \{\epsilon\} \). If we could decide that, then we could also decide ALLCFG by first converting the given \( G \) into a \( G' \) such that \( L(G') = L(G) \setminus \{\epsilon\} \), then converting \( G' \) into the NPDA \( M \) on which the hypothetical decision procedure \( A \) would be run.