(1) Consider the following three languages over the alphabet $\Sigma = \{a, b, c, d\}$, where by default $i, j, k, \ell$ are non-negative integers (can be 0):

$$L_1 = \{a^i b^j c^k d^\ell : i < j \land k < \ell\}$$

$$L_2 = \{a^i b^j c^k d^\ell : i < k \land j < \ell\}$$

$$L_3 = \{a^i b^j c^k d^\ell : i < \ell \land j < k\}.$$

One of these is not a CFL; the other two are CFLs. Give context-free grammars for the two that are CFLs, and a CFL Pumping Lemma proof for the one that is not a CFL. (You need not prove your grammars correct, but their plan should be clear. 6+6+18 = 30 pts.)

**Answer:** Intuitively, $L_1$ has “side-by-side dependencies,” $L_3$ has “nested dependencies,” but $L_2$ has “crossing dependencies”—where importantly these dependencies involve infinitely many values that must match. (Speaking again intuitively, all “finite dependencies” can be handled by a DFA.) Indeed, $L_1$ is the concatenation of $\{a^i b^j : i < j\}$ and $\{c^k d^\ell : i < j\}$. Here is a CFG for it:

$$S_1 \rightarrow T_1 T_2$$

$$T_1 \rightarrow aT_1 b \mid B$$

$$T_2 \rightarrow cT_2 d \mid D$$

$$B \rightarrow bB \mid b$$

$$D \rightarrow dD \mid D$$

There is also the shorter CFG with $T_1 \rightarrow aT_1 b \mid T_1 b \mid b$ and similarly for $T_2$—it avoids having variables $B$ and $D$ but maybe isn’t as clear. A CFG for $L_3$ can “re-use” the $D$ variable and use $C \rightarrow cC \mid c$ which similarly derives $c^+$:

$$S_3 \rightarrow aS_3 d \mid TD$$

$$T \rightarrow bTc \mid C$$

$$C \rightarrow cC \mid c, \quad D \rightarrow dD \mid d$$

To prove that $L_2$ is not a CFL, let any $p > 0$ be given (that is, you hear “$p$” from the adversary). The key is to take $s = a^p b^{p+1} c^{p+2} d^{p+3}$ (see note below on why $a^p b^{p+1} c^{p+2} d^{p+3}$ doesn’t work). This is a string that belongs to $L_2$ but “barely”—a “critical string.” Let any breakdown $s = uvxyz$ with $|vxy| \leq p$ and $vy \neq \epsilon$ be given. Then we get a set of mutually exhaustive cases according to whether the nonempty string $vy$ includes at least one (i) $a$, (ii) $b$, (iii) $c$, or (iv) $d$. (It doesn’t matter if two of these hold at once, but note that three cannot since $|vxy| \leq p$.)
(i) Take \( s^{(2)} = uv^2xy^2z \). It has at least \( p + 1 \) a’s, but still has exactly \( p + 1 \) c’s. It doesn’t matter whether \( w = s^{(2)} \) even still has the \( a^+b^+c^+d^+ \) form—it cannot obey the inequality \( \#a(w) < \#c(w) \) anymore, so \( s^{(2)} \not\in L_2 \).

(ii) We still take \( i = 2 \)—that is, we “pump up”—and take \( w = s^{(2)} = uv^2xy^2z \) again. This may not literally be the same string as in case (i) because the breakdown may be different, but the argument is similar: Being in this case means \( vy \) includes at least one \( b \), so \( w \) has at least \( p + 1 \) b’s, but it still has \( p + 1 \) d’s, so \( \#b(w) < \#d(w) \) is false. This implies \( s^{(2)} \not\in L_2 \).

(iii) Now if we “pump up” we will just make the inequality “more unequal” and stay in \( L_2 \), so we have to “pump down” with \( i = 0 \). Then \( w = s^{(0)} = uxz \) has at most \( p \) c’s, and it still has \( p \) a’s, so \( \#a(w) < \#c(w) \) fails, so \( w \not\in L_2 \) again, i.e., \( s^{(0)} \not\in L_2 \).

(iv) Again we pump down, making \( \#b(w) \geq \#d(w) \) this time, so \( s^{(0)} \not\in L_2 \). [I switched symbols to “w” only because writing \( \#b(s^{(0)}) \geq \#d(s^{(0)}) \) looks real ugly and it seems to have been Sipser’s intent to allow instructors to use the letter \( w \) for the pumped string.]

In all cases we have chosen \( i \) such that \( s^{(i)} \not\in L_2 \), and since the (adversary’s) breakdown was arbitrary, \( L_2 \) is a non-CFL by the CFL Pumping Lemma.

Footnotes: With \( s = a^pb^{p+1}c^p+2d^{p+3} \) the adversary can “survive” by giving the breakdown \( s = uvwxyz \) with \( v = c, x = \epsilon, y = d \) (and the “bookends” \( u = a^pb^{p+1}c^{p+1}, z = d^{p+2} \)). Then pumping down still leaves \( s^{(0)} = a^{p}b^{p+1}c^{p+1}d^{p+2} \) which still obeys the required inequalities for membership in \( L_2 \) even though the “middle one” no longer holds. So the adversary would get off scot-free in the hearing because of a bungled prosecution...

On an exam it would be perfectly fine to say that case (ii) is similar to (i) and (iv) is similar to (iii). It is deemed important, however, to list separate cases in full when one involves “pumping up” and the other involves “pumping down.”

(2) Let \( E \) be the language of nonempty strings of balanced parentheses. First show that the following CFG \( G \) is not comprehensive for \( E \) (that it is sound is pretty immediate so you need not prove it).

\[
S \to (S)S \mid ()
\]

Then add one or two more rules (that is, one or two more right-hand sides for \( S \)) to make a grammar \( G' \) that is comprehensive; presuming your rules are sound, you’ll get \( L(G') = E \) exactly. Then prove your answer in one of two ways:

- Prove \( E \subseteq L(G') \) by induction on strings (again we’ll regard the \( L(G') \subseteq E \) part as granted—so long as you made it true).

- Consider the grammar \( G'' \) given by \( S \to (S)S \mid \epsilon \). It is technically unsound because it derives \( \epsilon \), but it is well-known to be comprehensive. Then do some of the conversion of \( G'' \) to Chomsky normal form—how might this confirm your \( G' \) (or even give it to begin with)?

The latter option is quicker and makes this \( 3+6+9 = 18 \) pts. total, but the former will still be shown on the answer key.
Answer: The CFG $G$ does not derive either $(())$ or $(()($, both of which belong to $E$. This requires just noting that when you start with $S \to (S)S$ you’ve committed yourself to a string of length at least 6. So it is not comprehensive, so we need to add some rule(s) to make it so. (Further induction proofs are at the end.)

The Chomsky NF shortcut answer is to take as read that $G'' = S \to (S)S \mid \epsilon$ generates all balanced strings including $\epsilon$. The variable $S$ is nullable, so the first step toward Chomsky NF will give us a grammar $G_1$ such that $L(G_1) = L(G'') \setminus \{\epsilon\}$, which will give us $L(G_1) = E$ again. We have two occurrences of $S$ on the right-hand side of $S \to (S)S$, so we will get three new rules. Wiping out both occurrences gives us back $S \to (\cdot)$, which was our base rule in the original $G$, but wiping out one and not the other gives us $S \to (S)$ and $S \to (\cdot)S$. By the Chomsky normal form theorem, we’re done with $G' = G_1 = S \to (S)S \mid (S) \mid (\cdot)S \mid (\cdot)$.

(3) Design either a one-tape TM or a two-tape DTM $M$ such that $L(M) = \{a^ib^j : i < j, i \geq 0\}$. If you do the latter—which not only runs more quickly but also has cleaner code IMHO—make it obey the condition of being a deterministic pushdown automaton: no character changes or left-moves on the input tape, and any left-move on the second tape must write the blank. A well-commented arc-node diagram is expected—not (just) a pseudocode strategy. It is OK to use the text’s diagram style, but especially if you choose to do the 2-tape TM (which a DPDA “Is-A”) you may find my “stacked instruction labels” have a more-uniform look-and-feel. (15 pts., for 63 total on the set)

Answer in prose: Both the one-tape TM and the two-tape TM can use their start state $s$ already to start checking an acceptance condition. If the TM reads a $b$ in $s$, then it goes to a “pre-final” state $p$ that loops to itself on $b$, goes to $q_{\text{rej}}$ if an $a$ comes along that ruins the $a^*b^*$ format, and goes to $q_{\text{acc}}$ upon reading either the blank $B$ or a special $-$—whichever convention you use to mark the end of the input string $x$. If instead the TM reads an $a$ then we part ways. The one-tape TM overwrites the $a$ by $X$ and goes to a state $q$ that self-loops over any more $a$’s (and any more $X$’s) until it finds a $b$—if it doesn’t then a $B$ or $-$ goes to $q_{\text{rej}}$. On $b$ it goes to a third state $r$ which self-loops moving leftward until it hits an $X$ that was earlier written (or hits the $B$ to the left of $x$, or hits a left-endmarker $\wedge$ if you use that convention). Whatever it hits, it steps right and goes back to $s$.

As a meaningful footnote for strategy, the two-tape TM on reading an $a$ instead executes a transition to a state $q$ that copies $a$ to tape 2 (rather than overwriting it on tape 1). In state $q$ it keeps similarly copying $a$’s until on a $b$ it momentarily pauses by staying stationary on the input tape and moving its tape-2 head back left to enter “pop mode.” Thereafter it keeps popping so long as it sees $b$ on tape 1 (moving one step right each time) and $a$ on tape 2 (popping by writing $B$ on it and moving left—this complies with the tape-2 restriction for PDAs). If it runs out of $a$’s on tape 2, it keeps reading $b$’s just to make sure there are no trailing $a$’s which would ruin the $a^*b^*$ format and then accepts. If it hits the end of the input tape (whether marked by $B$ or by $-$) without having already run out of $as$, it rejects—even if it had just popped the last one that means $j = i$ which doesn’t work.

The 1-tape TM does one pass for each $a$ and hence runs in worst-case $O(n^2)$ time. The two-tape TM does just one pass and runs in $O(n)$ time.

Grammar Footnotes on Problem (2): As sketched on Piazza, by taking $P_S = “Every \, x \, I \, derive \, has \, |x| \equiv 2 \, \text{modulo} \, 4,”$ you get that this is witnessed by the base rule $S \to (\cdot)$ and
preserved by the other rule \( S \rightarrow (S) \). This is because if \( S \rightarrow^* x \) using the latter rule first, then \( x = (y)z \) where \( S \rightarrow^* y \) and \( S \rightarrow^* z \). By IH \( P_S \) on the right-hand side (twice), \(|y| \equiv 2\) and \(|z| \equiv 2\) modulo 4. Hence \(|x| \equiv 1 + 2 + 1 + 2 \equiv 6 \equiv 2\) modulo 4, which upholds \( P_S \) on the left-hand side. So the original \( G \) does not derive any strings of lengths 4, 8, 12, 16, \ldots either.

The long way to show the comprehensiveness of the grammar \( G' \) augmented with the rules \( S \rightarrow ()S | (S) \) alongside the original \( S \rightarrow (S)S | () \) is to prove \((\forall n \geq 2)P(n)\) where \( P(n) \equiv \) “for each \( x \) of length \( n \), if \( x \in E \) then \( S \rightarrow^* x \)”.

Note that this implication is true by default for odd \( n \), so we are only concerned with even \( n \). And we don’t have \( P(0) \) as our base case, only \( P(2) \), which we should prove first:

\( P(2) \): The only \( x \) of length 2 that belongs to \( E \) is \( x = () \). And \( S \rightarrow x \) immediately. So \( P(2) \) holds.

**Induction \((n > 2)\):** We may assume as induction hypotheses (IH) the statements \( P(m) \) for any \( m \), \( 2 \leq m < n \), but not \( m = 0 \). Let any string \( x \in E \) with \(|x| = n \) be given. Then \( x \) must begin with ‘(’ . Numbering strings from 1, let \( i > 1 \) be the index of the ‘)’ that is the “matching mate” to the initial ‘(’ . In terms of the \( \text{diff}(x, i) \) function—which in lecture was defined as \( \#a(x) - \#b(x) \) but here we have ‘(’ in place of ‘a’ and ‘)’ in place of ‘b’—the matching mate is defined by the least \( i > 1 \) such that \( \text{diff}(x, i) = 0 \). This gives us a unique breakdown \( x = (y)z \) such that \( y \) and \( z \) are balanced—but either could be empty and so not derivable from \( S \) in \( G' \). That’s why we need cases and the other rules.

**Case \((i)\):** \( y \) is empty. Then \( x = ()z \) where \(|z| = n - 2 \) and \( z \neq \epsilon \) (since \( n > 2 \)). By IH \( P(n - 2) \), \( S \rightarrow^* z \). So we derive \( S \rightarrow ()S \rightarrow^* ()z = x \).

**Case \((ii)\):** \( z \) is empty. Then \( x = (y) \) where \(|y| = n - 2 \) and \( y \neq \epsilon \) (again since \( n > 2 \)). By IH \( P(n - 2) \), \( S \rightarrow^* y \). So \( S \rightarrow (S) \rightarrow^* (y) = x \).

**Case \((iii)\):** neither \( y \) nor \( z \) is empty. Then \( m_1 = |y| \) and \( m_2 = |z| \) are both “in-bounds” for our induction hypotheses: both \( \geq 2 \) and both \( \leq n - 2 \). So by IH \( P(m_1) \) and \( P(m_2) \), \( S \rightarrow^* y \) and \( S \rightarrow^* z \). Thus \( S \rightarrow (S)S \rightarrow^* (y)S \rightarrow^* (y)z = x \).

Thus in all three of these mutually exhaustive cases we get \( S \rightarrow^* x \). Since \( x \) was an arbitrary length-\( n \) member of \( E \), \( P(n) \) follows, so \((\forall n \geq 2)P(n)\) is shown by strong induction. This finally says \( E \subseteq L(G') \). Since \( G' \) is clearly sound for \( E \), \( L(G') = E \).