Consider the context-free grammar $G = (V, \Sigma, R, S)$ with $\Sigma = \{a, b\}$ and rules

- $S \rightarrow TU$
- $T \rightarrow aTa \mid bTb \mid U$
- $U \rightarrow aU \mid bU \mid \epsilon$

(a) Create a grammar $G'$ such that $L(G') = L(G) \setminus \{\epsilon\}$ and $G'$ has no $\epsilon$-rules. Along the way, say which variables of $G$ are nullable.

(b) Show that $G$ is ambiguous by giving two different parse trees for some string $x \in L(G)$.

(c) Is $G$ comprehensive? “Comprehensive for what language?”, you may ask. Does it matter?

Answer: (a) First $U$ is nullable, then the next round catches $T$, then the next round catches $S$ from $S \rightarrow TU$. Adjusting gives $G' =$

- $S \rightarrow TU \mid T \mid U$
- $T \rightarrow aTa \mid bTb \mid U \mid aa \mid bb$
- $U \rightarrow aU \mid bU \mid a \mid b.$

(b) There are lots of answers but it pops out that $T$ derives $aa$ which can also be derived from $UU$. Both trees start out with $S \rightarrow TU$ from the root, but one does $T$ to $aTa$ and “epsilon-out” the $T$ and $U$, whereas the other does $T \rightarrow U$ and both $U$s to $aU$ then $a$.

(c) The variable $U$ derives all strings and $S \Rightarrow TU \Rightarrow UU \Rightarrow U$. So $S$ too derives all strings, so $L(G) = \Sigma^*$, which is automatically comprehensive for any language. So the answer (with this explanation) is ‘yes’ and ‘it doesn’t matter.’

Footnote: In any derivation, $T$ will derive tut$^R$ where the $u$ comes from $U$. The “intent” of the variable $S$ is thus to do $\{tut^Ru': t, u, u' \in \Sigma^*\}$. If we group the $utu'$ and call it $x$, then the intent is to represent $\{tx : t^R$ is a substring of $x\}$. The catch is that $t = \epsilon$ is allowed here and makes the language trivial, so we get any $x$ which is why the language is $\Sigma^*$. But we can fix this by changing the rules of $U$ to $U \rightarrow Ua \mid Ub \mid \#$.

(2) (12 + 12 + 6 = 30 pts. total)

Recall the definition of $\#u(x)$ as the number of times the substring $u$ occurs within the string $x$, counting overlaps—so that for instance $\#aa(baaab) = 2$. Define $T = \{x : \#aa(x) = \#bb(x)\}$. Let $G = (V, \Sigma, R, S)$ be the context-free grammar with variables $V = \{S, A, B\}$ and rules

- $S \rightarrow BSA \mid \epsilon$
- $A \rightarrow Aab \mid aab$
- $B \rightarrow baB \mid bb$
(a) Consider these four “target properties”: (i) “begins with a,” (ii) “ends with b,” (iii) “begins with \(a\),” and (iv) “ends with \(b\).” Say which one(s) hold for strings derived from \(A\), and which one(s) hold for strings derived from \(B\).

(b) Use your targets in (a), and other facts you may add to them, to show that \(L(G) \subseteq T\).

(c) Suppose \(G'\) adds the rule \(S \rightarrow ASB\) to \(G\). Is \(G'\) still sound, i.e., is \(L(G') \subseteq T\)? If you say yes, prove it; if you say no, give a leftmost derivation of a string \(x \in L(G') \setminus T\).

Answer: (a) The variable \(A\) enforces (i) and (iv), while \(B\) enforces (iii) and (iv) (as intended) without the fix, (ii) and (iii) with it.

(b) Take the target language \(T_A\) for \(A\) to be the set of strings that begin with \(a\), end with \(b\), and have exactly one \(aa\) (at the beginning). Take \(T_B\) for \(B\) to be the strings that begin and end with \(b\) and have exactly one \(bb\) (at the end). The extra “one \(aa\)” and “one \(bb\)” properties are also maintained (indeed, the exact language of \(A\) is \(a(ab)^+\) and that of \(B\) is \((ba)^*bb\)—this was originally part of the problem). The rule \(S \rightarrow \epsilon\) satisfies the “0 = 0” clause of \(T\). Hence it remains to verify the rule \(S \rightarrow BSA\). This rule preserves the “\(#aa(x) = #bb(x)\)” balance within the substrings derived from \(A\) and \(B\), so it remains to check the junctures.

The intended point was that when \(S\) eventually disappears via \(S \rightarrow \epsilon\) you have \(BA\), but that juncture does not add any \(aa\) or \(bb\) because the part derived from the \(B\) ends in \(b\) and the part derived from the \(A\) begins with \(a\). The bug however is that the \(BS\) juncture can also become \(BB\), and the begins-and-ends-with-\(b\) conjunction of properties (iii) and (iv) creates an extra \(bb\). Namely, \(S \Rightarrow BSA \Rightarrow bbSA \Rightarrow bBB\) \(\Rightarrow bbbb\) \(\Rightarrow bb\) \(\Rightarrow bbbb\) \(\Rightarrow bbbb\) \(\Rightarrow bbbbaab\) \(\Rightarrow bbbbaabaab\), but \(x = bbbbaabaab \notin T\) because \(#bb(x) = 3\) versus \(#aa(x) = 2\). (The juncture with \(AA\) is fine.)

Attempting to patch this by replacing \(B \rightarrow bb\) with \(B \rightarrow bba\) makes \(BA\) bad, however. The problem is not easy to patch and keep \(S \rightarrow BSA\) good but \(S \rightarrow ASB\) bad. Doing \(S \rightarrow aBSA\) works but is not so neat.

(c) The bug—that is, the intended bug—is that \(S \rightarrow \epsilon\) can (and must) create an extra \(bb\) in the middle. An illustrative leftmost derivation is \(S \rightarrow ASB \rightarrow aabSB \rightarrow aabB \rightarrow aabb\).

\[3\] \(18 + 15 + 4 = 37\) pts.

Define \(L = \{ x \in \{ a, b \}^* : x = x^R \land \#a(x) = \#b(x) \}\).

(a) Prove using the CFL Pumping Lemma that \(L\) is not a context-free language. (Hint: Consider strings in \(L\) that also belong to \(a^*(bb)^*a^*\).)

(b) Sketch in prose a 2-tape deterministic Turing machine \(M\) such that \(L(M) = L\). Describe the operation of \(M\) as a finite sequence of “passes” and say what happens in each pass. Is your \(M\) a pushdown automaton? Could it be one?

Answer (a) Given a pumping length \(p > 0\), choose \(s = a^p b^p d^p\). Then \(s \in L\). Let any breakdown \(s = yuvwz\) with \(uw \neq \epsilon\) and \(|uvw| \leq p\) be given. We break into cases according to whether \(uw\) includes any \(a\)’s. If yes, then by \(|uvw| \leq p\), it can contain them from only one of the two \(a^p\) blocks. Hence pumping down to \(s_0 = yvz\) unbalances the blocks. Since doing so cannot also remove all the \(b\)’s, the resulting string has different-length strings of maximal consecutive \(a\)’s at its ends, hence cannot be a palindrome, so \(s_0 \notin L\). Else, it contains only \(b\)’s. Hence pumping down creates \(a^p b^q a^p\) where \(q < 2p\), so it violates the condition \(#a(x) = #b(x)\), so again \(s_0 \notin L\). Hence \(L\) is not a CFL, by the CFL Pumping Lemma.
(b) It is not necessary to find the “halfway” point in \( x \) but there was nothing wrong with saying how to do so (by passing over \( x \) on tape 1 and moving the tape-2 head one step for every two steps by the tape-1 head, having first written a \( \wedge \) left-endmarker on tape 2 so as to know when to stop copying the right-half of \( x \) underneath the left-half). Here are the passes without doing so:

1. Copy \( x \) to tape 2 (use of left endmarker good but actually optional now).

2. Rewind either head to the left end. (If the tape-1 head then this violates the PDA condition of its not moving left; if the tape-2 head then you are moving left without popping which violates the other condition.)

3. Make the heads do a bit-by-bit comparison of \( x \) with \( x^R \) by crossing each other on the tape. If a mismatch is found, reject (after an optional cleanup phase).

4. Blank out Tape 2 (if not already done); it doesn’t matter which end the tape-1 head is at.

5. Pass over \( x \) on tape 1 and push an \( A \) on tape 2 for every \( a \) read.

6. One more pass over \( x \) pops an \( A \) for every \( b \) read and accepts if and only if tape 2 empties at the same time tape 1 has no more \( b \)'s. (Use of \( \wedge \) and $ endmarkers helps to visualize this but is optional here; note that tape 2 will be automatically cleaned up and the tape-1 head could wind up at the end of \( x \) if we care about that clause of “good housekeeping” too.)

It was also find to do the \(#a(x) = #b(x)\) comparison first, but either way the second comparison then violates the “can’t rewind tape 1” PDA condition at least. It is impossible to design a PDA because the language is not a CFL. (Incidentally, the 2-tape code runs in \( O(n) \) time where \( n = |x| \); it is known but not covered in our text that every 1-tape TM recognizing this language requires order-of \( n^2 \) time.)

(4) (15 pts.)

Define \( r = b(aa)^*b \). Consider \( S_0 = \{ \epsilon, a, b, ba \} \). This is a PD set for \( L(r) \), but it is not maximal. Find another string \( u \) such that \( S = S_0 \cup \{ u \} \) is a PD set of size 5 for \( L(r) \). Show your work to verify this; drawing a DFA is optional and might be beside the point.

Answer: This problem has a shortcut way to find \( u \): All four of the given strings \( \epsilon, a, b, ba \) do not belong to \( L(r) \). Hence we need only find a string \( u \) that does belong to \( L(r) \) and it will be automatically distinguished from these four strings by “\( z = \epsilon \)” Such a string is \( u = bb \), or alternatively \( u = baab \) (etc.).

Even if this point was missed, partial credit was given for distinguishing the four given strings from each other and/or building a correct DFA. Here is the whole table of the five strings:

<table>
<thead>
<tr>
<th>( z )</th>
<th>( \epsilon )</th>
<th>( a )</th>
<th>( b )</th>
<th>( ba )</th>
<th>( bb )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \epsilon )</td>
<td>( bb )</td>
<td>( b )</td>
<td>( ab )</td>
<td>( \epsilon )</td>
<td></td>
</tr>
<tr>
<td>( a )</td>
<td>( b )</td>
<td>( ab )</td>
<td>( \epsilon )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( b )</td>
<td>( b )</td>
<td>( \epsilon )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( ba )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>