(1) Let $G = (\{ I \}, \{ s, d \}, \mathcal{P}, I)$ be the context-free grammar with rules $I \rightarrow sI | sIdI | \epsilon$. Let $T$ be the language of strings $x$ over $\{ s, d \}$ such that for every prefix $y$ of $x$, $\#s(y) \geq \#d(y)$. For a fact, $L(G) = T$—you are not asked to prove this. For further interpretation, note that if you interpret $s = \text{“spear”}$ and $d = \text{“dragon,”}$ then $T$ specifies the strings in which you “survive if you can hold arbitrarily many spears.” More prosaically, if you interpret $s$ as a left-paren and $d$ as a right-paren, then $T$ becomes the language of strings that might not yet be balanced, but can be closed out to be balanced by appending some number of $’)$ parens.

(a) Give both a parse tree and a leftmost derivation for each of the following strings in $T$ (3 × 6 = 18 pts.):

(i) $x_1 = sdssd$
(ii) $x_2 = sssddsdd$
(iii) $x_3 = ssddsdsdd$.

(b) Show that $G$ is ambiguous, by finding an ambiguous string in $T$ and giving two distinct derivation trees or two distinct leftmost derivations—your choice. (There are even shorter strings than the above. 9 pts., for 27 on the problem.)

**Answer:** (a) The basic idea is that when you know which $s$ “kills” which $d$, you use the $I \rightarrow sIdI$ rule to derive them. Now for (i) $x = sdssd$, you know the first spear and dragon are paired, so you begin the leftmost derivation with $I \Rightarrow sIdI \Rightarrow sdI$. Then we’re left with needing to do $ssd$. Either $s$ could kill the $d$, so we can continue with either rule. So either:

\[
I \quad \Rightarrow \quad sIdI \quad \Rightarrow \quad sI \quad \Rightarrow \quad sdsI \quad \Rightarrow \quad sdssI \quad \Rightarrow \quad sdssIdI \quad \Rightarrow \quad sdssd;
\]

\[
I \quad \Rightarrow \quad sIdI \quad \Rightarrow \quad sI \quad \Rightarrow \quad sdsIdI \quad \Rightarrow \quad sdssIdI \quad \Rightarrow \quad ^2 sdssd.
\]

(ii) This $x$ has 4 spears and 4 dragons, so we never have the luxury of using the rule $I \rightarrow sI$. We have to figure out which $I$s in the right-hand side of $I|tosIdI$ need to be “epsilon”-ed and which need to be expanded to place a spear early enough. Working it out gives:

\[
I \quad \Rightarrow \quad sIdI \quad \Rightarrow \quad s sIdI \quad \Rightarrow \quad s s sIdI \quad \Rightarrow \quad sssIdIdI \quad \Rightarrow \quad ssddsIdIIdI \quad \Rightarrow \quad ^3 sssddsdd = x.
\]

(iii) $x = ssddsdsdd$. Break first as $ssdd \cdot ssddsdd = s \cdot sd \cdot d \cdot ssddsdd$. At the start we need to begin $I \Rightarrow sIdI \Rightarrow ssIdIdI \Rightarrow ssIdI \Rightarrow ssddI$. Now we need to get $ssddsdd$ from the final $I$, and that goes:

\[
ssddsI \quad \Rightarrow \quad ssddsIdI \quad \Rightarrow \quad ssddsIdI \quad \Rightarrow \quad ssddsIdIIdI \quad \Rightarrow \quad ^2 ssddsdd.
\]

The parse trees are uniquely defined by these derivations.

(b) We’ve already seen the ambiguous string $ssd$ in (a)(i), which makes $sdssd$ ambiguous too with the two LM derivations shown. Just for $ssd$, we have:
I ==> sI ==> ssIdI ==> ssdI ==> ssd (first sword ignored)
I ==> sIdI ==> ssIdI ==> ssdI ==> ssd (first sword used)

(2) Let $G = (\{S, A, B, C, D\}, \Sigma, R, S)$ be the context-free grammar with $\Sigma = \{a, b\}$ and rules $R =$

$$S \rightarrow AD | bbC | SaBS$$
$$A \rightarrow BAB | \epsilon$$
$$B \rightarrow SB | b$$
$$C \rightarrow ACD | BA | DAS$$
$$D \rightarrow BaaC | \epsilon.$$ 

(Since the period is not in $\Sigma$, it is just punctuation.)

Find a grammar $G'$ without $\epsilon$-rules such that $L(G') = L(G) \setminus \{\epsilon\}$. Show clearly which variables in $G$ are nullable. Is $\epsilon \in L(G)$? It is OK if your final $G'$ looks “gross” so long as you show the steps of the algorithm clearly. (18 pts.)

**Answer:**

(1) $S \rightarrow AD | bbC | SaBS$
$A \rightarrow BAB | \epsilon$
$B \rightarrow SB | b$
$C \rightarrow ACD | BA | DAS$
$D \rightarrow BaaC | \epsilon.$

A and D are immediately nullable, then so is S by the rule $S \rightarrow AD$. That in turn makes C nullable, by the rule $C \rightarrow DAS$. However, B remains non-nullable, since (by SI) every string it derives has at least one 'b'. Modified grammar, in BNF form—every variable other than B is optional:

$$S \rightarrow [A][D] | bb[C] | [S]aB[S]$$
$$A \rightarrow B[A]B | \epsilon$$
$$B \rightarrow [S]B | b$$
$$C \rightarrow [A][C][D] | B[A] | [D][A][S]$$
$$D \rightarrow BaaC | \epsilon.$$ 

Conversion back to CFG, w/o writing new epsilon-rules and deleting old ones:

$$S \rightarrow AD | A | D | bbC | bb | SaBS | aBS | SaB | aB$$
$$A \rightarrow BAB | BB$$
$$B \rightarrow SB | B | b$$
$$C \rightarrow ACD | AC | AD | CD | A | C | D | BA | B | DAS | DA | DS | AS | D | A | S$$
$$D \rightarrow BaaC | Baa.$$

Since $S$ is nullable, yes \(\epsilon\) is in $L(G)$, but this new grammar $G'$ gives $L(G') = L(G) \setminus \{\epsilon\}$. Note that $G'$ can be cleaned up by deleting redundant "unit rules" including "$B \rightarrow B".$
(3) Let \( G = (\{ S, A \}, \Sigma, \mathcal{R}, S) \) be the context-free grammar with \( \Sigma = \{ a, b \} \) and rules \( \mathcal{R} = \)

\[
\begin{align*}
S & \rightarrow SS \mid ASa \mid aA, \\
A & \rightarrow bA \mid SAa \mid a.
\end{align*}
\]

And let \( T \) be the language of strings \( x \) such that \( \# \) \( a(x) \) is even.

(a) Show that this grammar is unsound for \( T \)—that is, find a string in \( L(G) \setminus T \). (6 pts.)

(b) Change one of the rules to make a grammar \( G' \) that is sound for \( T \). Make it a “single-edit” change: that is, inserting, deleting, or changing just one character (terminal or variable symbol). Then explain why your new \( G' \) is sound. (12 pts. total)

(c) Is the original grammar comprehensive for \( T \)? OK, it’s not: \( \epsilon \in T \setminus L(G) \). So let’s change the target language to be \( T' = T \setminus \{ \epsilon \} \) and revise the question: Is the original grammar comprehensive for \( T' \)? If you say yes, argue why as best you can; if you say no, give a nonempty string with an even number of \( a \)'s that \( G \) does not generate, and explain why not. (12 pts., for 30 on this problem and 75 on the set)

Answer: (a) \( S \rightarrow aA \rightarrow aSAa \rightarrow aaAAa \rightarrow 2 aaaaaa \), but \( aaaaaa \notin T \), so \( L(G) \not\subseteq T \), so \( G \) is unsound. (There are also derivations of \( aaaaaaa \) and so on.)

(b) In my opinion, the neatest fix is to change \( SAa \) to \( SAb \), or just to \( SA \). Doing the latter gives,

\[
\begin{align*}
S & \rightarrow SS \mid ASa \mid aA, \\
A & \rightarrow bA \mid SA \mid a.
\end{align*}
\]

To explain why it is sound, think of \( S \) as signifying “even” and \( A \) as “odd,” also \( a \) adds 1 and \( b \) adds 0. The right-hand sides of the rules for \( S \) say, respectively, “even + even,” “odd + even + 1,” and “1 + odd”; these all add up to “even” and this preserves the meaning of \( S \) as “even.” Whereas, the right-hand sides of \( A \) say: “0 + odd,” “even + odd,” and “1”—all of which are odd and so preserve the soundness of \( A \) meaning “odd.” This mutual preservation—checking each rule of each variable—is enough to constitute a proof that the whole grammar is sound.

(c) \( G \) is not comprehensive. It does not derive simply \( b \), nor \( aab \), and indeed derives only strings that end in \( a \). To see this “at-a-glance,” augment the properties of \( S \) and \( A \) above to include a clause that every string they derive ends in \( a \). Each right-hand side of each rule either has a literal \( a \) or it has a variable, which preserves these clauses in every derivation.