(1) Consider the following part of a grammar for expressions in Java.

E  -->  E2 ASSGTOP E | E2  //assignment is right-associative
E2  -->  E2 BINOP E3 | E3  //BINOPs are left-associative
E3  -->  +E3 | -E3 | ++P | --P | E4
E4  -->  P++ | P-- | P
P  -->  (E) | LITERAL | VARIABLE (etc.)
ASSGTOP  -->  = | += | -= (etc.)
BINOP  -->  == | != | + | - | * | / (etc.)
LITERAL  -->  /any number or quoted string etc./
VARIABLE  -->  /any legal identifier/.

Important for this problem is the supposition that any binary + and - has whitespace separating it from any nearby + or - signs in your program text—this is necessary for proper lexing of your program. Unary +, - and the pre- and post-increment operators do not need such whitespace, but they still need separation from each other; e.g. \(-(-a)\) separate from \(--a\). [For a footnote, programming languages resolve the potential ambiguity of whether e.g. \("--a\) is parsed as \(--(-a)\) or as pre-decrement, by using the rule that the longest legal token is selected. Since \(--\) is a longer token than two separate negation signs, \("--a\) is lexed as pre-decrement. This is not considered an ambiguity in the grammar. In fact, you can do this problem treating ++ and -- as single separate symbols, and then the grammar is formally unambiguous.]

(a) Show how to derive a legal Java expression that has the substring "++ + ++" in it, noting the whitespace around the binary +. (6 pts.)

(b) Prove by “structural induction” that the substring "+ ++ +" can never occur in a legal Java expression, nor any similar one with - in place of + and/or -- in place of ++. (15 pts.)

Answer: (a) To derive e.g. \(x++ + ++y\), do the following, which is in fact perfectly legal in C/C++/Java/C#…

\[
E \rightarrow E2 \rightarrow E2 \text{BINOP} E3 \rightarrow E3 \text{BINOP} E3 \\
\rightarrow E4 \text{BINOP} E3 \rightarrow P++ \text{BINOP} E3 \rightarrow \sim^2 x++ \text{BINOP} E3 \rightarrow x++ + E3 \\
\rightarrow x++ + ++P \rightarrow \sim^2 x++ + ++y
\]

(b) Show that the sequence of tokens + ++ + cannot be derived. Assign to the grammar nonterminal \(P\) the meaning, “Every string \(x\) I derive begins and ends with a parenthesis, alphanumeric char, or quotes.” This is immediately preserved by the productions for \(P\) (as was clarified after student queries about literals). Then we can assign to all of \(E, E2, E3, E4, P\) the meaning, “Any ++ or -- token in a string I derive is immediately preceded or followed by a parenthesis, alphanumeric char, or quotes.” Note that for \(P\) this adds to the previous meaning–it is needed to “inherit” this to \(P\) because \(P\) can derive \((E)\). Then the rules preserve this latter meaning too—immediately in the case of right-hand sides ++P, --P, P++, P-- because of the meaning of \(P\), and by induction for the other right-hand sides. The most sensitive cases are where \(E3\) is preceded by a unary + or - or BINOP on the right-hand side. Then the point is that any ++ or -- derived by the \(E3\) on the right-and side (and by the \(E2\) preceding the \(\text{BINOP}\)) is going to have a non-operator precede or follow it. We don’t have to worry about the words “it isn’t preceded by an operator” allowing +E3 to violate because we “accentuated the positive”—we said any ++ or -- “in” \(E3\) is positively either preceded or followed by a non-operator, which takes care of anything the + in +E3 could threaten. Ditto with either side of BINOP or the front side of ASSGTOP.
(2) Consider the grammar $G$ of Problem 2.19 on page 156 (2nd ed., p130; in the 1st ed., it’s problem 2.25):

\[ S \rightarrow aSb \mid bY \mid Ya, \quad Y \rightarrow bY \mid aY \mid \epsilon. \]

Suppose we answer the text’s first question by saying, “$G$ generates the language $E$ of all nonempty strings over the alphabet \{a, b\}.” This answer has a tiny bug. Fix the bug by adding a rule with no variables on the right-hand side, and prove that the resulting grammar $G'$ is correct. You may assert without further proof that $Y$ derives all strings, and also note that $L(G') \subseteq E$ is immediate since $S$ does not derive $\epsilon$. Hence you only need to handle the $S$ cases in proving $E \subseteq L(G')$ (12 pts)

**Answer:** Simply because $S$ is not nullable, the grammar $G$ is immediately sound for $E$, i.e. $L(G) \subseteq E$. Is it equal? No—the grammar does not derive ab. Let’s see how we might discover both this bug and the fix in the act of trying to prove comprehensiveness.

We can take for granted that $Y$ derives every string, so we need only focus on $S$, in proving

\[ (\forall n \geq 1)Q(n), \quad \text{where } Q(n) = \text{if } |x| = n \text{ then } S \Rightarrow^* x. \]

**Basis** ($n = 1$, not $n = 0$): The only strings $x$ of length 1 are $x = a$ and $x = b$. Then we can do $S \Rightarrow Ya \Rightarrow a$ and $S \Rightarrow bY \Rightarrow b$, both since $Y \Rightarrow \epsilon$. So the basis holds.

**Induction** ($n \geq 2$). We may assume (IH) the statement $Q(m)$ for all $m < n$—except that is for $m = 0$. Let any string $x$ of length $n$ be given. Since $x \neq \epsilon$, one of the following ”parsing cases” must hold: either

(i) $x$ begins with $b$,

(ii) $x$ ends with $a$, or

(iii) $x$ begins with $a$ and ends with $b$.

In case (i), $x = by$ for some string $y$. Since we’ve already observed that $Y$ can derive any string $y$, we have $S \Rightarrow bY \Rightarrow^* by = x$, so $S \Rightarrow^* x$. (Again, we didn’t need to define a requirement language $L_Y$ for the variable $Y$ because we already know $Y$ derives any string.)

In case (ii), $x = ya$ for some string $y$, and we get $S \Rightarrow Ya \Rightarrow^* ya = x$, so $S \Rightarrow^* x$.

In case (iii), $x = azb$ for some string $z$. Since $|z| = n - 2 < n$, we may apply IH $Q(n - 2)$ to deduce that $S \Rightarrow^* z$. This gives $S \Rightarrow^* aSb \Rightarrow^* azb = x$.

Proof done? It looks like we got $S \Rightarrow^* x$ in all three cases. Since these cases are exhaustive for strings $x$ of length $n$ at least 2, it looks like we can conclude $Q(n)$, thus making “$(\forall n Q(n))$” follow by induction, which formally gives us $E \subseteq L(G)$ so that $E$ and $L(G)$ are equal...

Well not so fast! We already noted the bug that $S$ does not derive ab, which ahs length $n = 2$. So what goes wrong in the induction case for $n = 2$? It may remind you of the fallacy in the induction “proof” discussed in week 1 that all horses have the same color, which also broke down in the $n = 2$ case though for different reasons. Here the gap is that when $n = 2$, the quantity $n - 2$ equals 0, and the “IH” is $Q(0)$—except we don’t have $Q(0)$ available since we based at $Q(1)$.

Moreover, the strings aabb, aaabbb, aaaaabbb, etc., aren’t in $L(G)$ either. However, the proof goes through fine if we can treat $n = 2$ as “another base case” upon adding the rule $S \rightarrow ab$, which meets the terms fo the problem. Then we have $n \geq 3$ in the induction and everything works.
(3) Let \( E = \{ x \in \{ a, b \}^* : \#b(x) - \#a(x) \text{ is a multiple of 3} \} \), where as usual \( \#a(x) \) means the number of \( a \)'s in the string \( x \). Let \( G \) be the context-free grammar

\[
S \rightarrow aA \mid SbB \mid AB \mid \epsilon \\
A \rightarrow aa \mid bS \mid aB \\
B \rightarrow SaS \mid AA.
\]

(a) Prove that \( L(G) = E \). (9+18 = 27 pts. Reasonable proof shortcuts are OK.)

(b) Find (at least) two productions that you can delete from this grammar without changing the language it generates. Justify your answer by explaining why those rules aren’t needed in your proof of \( E \subseteq L(G) \) in part (a).

**Answer:** For soundness, i.e., to prove \( L(G) \subseteq E \), define the following properties:

- \( P_S = \) “All \( x \) I derive have \( \#b(x) - \#a(x) = 0 \) mod 3.”
- \( P_A = \) “All \( x \) I derive have \( \#b(x) - \#a(x) = 1 \) mod 3.”
- \( P_B = \) “All \( x \) I derive have \( \#b(x) - \#a(x) = 2 \) mod 3.”

Note: This requires that \(-1 = 2 \) mod 3, as opposed to being \(-1 \) mod 3. The former is mathematically correct, but “mod” in some programming languages behaves the latter way! Now to verify the rules:

- \( S \rightarrow \epsilon \): Since \( 0 - 0 = 0 \) mod 3, this upholds \( P_S \) on LHS.
- \( S \rightarrow SbB \): Suppose \( S \Rightarrow^* x \) utrf. Then \( x = ybz \) where \( S \Rightarrow^* y \) and \( B \Rightarrow^* z \). By IH \( P_S \) on RHS, \( y \) contributes 0 to \( \#b(x) - \#a(x) \), while \( z \) contributes +2. The ‘\( b \)’ makes that +3, which = 0 mod 3, upholding \( P_S \) on LHS.
- \( A \rightarrow aB \): Suppose \( A \Rightarrow^* x \) utrf. Then \( x = ay \) where \( B \Rightarrow^* y \). By IH \( P_B \) on RHS, \( \#b(y) - \#a(y) = 2 \) mod 3. The initial ‘\( a \)’ subtracts 1 from this, giving \( \#b(x) - \#a(x) = 1 \) mod 3, upholding \( P_A \) on LHS.
- The other rules are “similar” in applying the counts: \( S = 0 \), \( A = +1 \), ‘\( a \)' = -1, \( B = +2 \), ‘\( b \)' = +1. The right-hand sides of the rules for \( S \) add up to 0 or 3, those for \( A \) to -2 (which = 1 mod 3) or +1, and those for \( B \) to -1 or +2.

For comprehensiveness, i.e. \( E \subseteq L(G) \), we need to prove for all \( n \geq 0 \) the statement

- \( P(n) \) == “for each \( x \in a, b^n \): if \( \#b(x) - \#a(x) = 0 \) mod 3 then \( S \Rightarrow^* x \).”

Augment this to proving \((\forall n)P'(n)\), where \( P'(n) = P(n) \wedge Q(n) \wedge R(n) \), and

- \( Q(n) \) == “for each \( x \in a, b^n \): if \( \#b(x) - \#a(x) = 1 \) mod 3 then \( A \Rightarrow^* x \).”
- \( R(n) \) == “for each \( x \in a, b^n \): if \( \#b(x) - \#a(x) = 2 \) mod 3 then \( B \Rightarrow^* x \).”
Basis \((n = 0)\): The only such \(x\) is \(\epsilon\). It gives difference 0, and \(S \Rightarrow^* \epsilon\) is immediate by a rule. So this checks out.

Induction \((n \geq 1)\): Assume \((\text{IH})\) \(P'(m)\) for all \(m < n\), i.e., \(P(m) \land Q(m) \land R(m)\). Goal: show \(P'(n)\). Let any \(x \in a, b^m\) be given. Break into cases and subcases:

\((P)\) \#\(b(x)\) − \#\(a(x)\) = 0 mod 3. Since \(x \neq \epsilon\), we have two subcases:

(i) \(x\) begins with \(a\). Then \(x = ay\) for some string \(y\) of length \(n - 1\), and necessarily \#\(b(y)\) − \#\(a(y)\) = 1 mod 3, so that subtracting 1 for the \(a\) in front leaves 0 mod 3. Since \(|y| = n - 1 < n\), we may apply \(Q(n - 1)\) from \(\text{IH}\) to get \(A \Rightarrow^* y\). Then we build the derivation \(S \Rightarrow aA \Rightarrow^* ay = x\), showing \(P(n)\) in this subcase, which is what we need for \(P'(n)\).

(ii) \(x\) begins with \(b\). Then \(x = by\) where \#\(b(y)\) − \#\(a(y)\) = 2 mod 3. By \(\text{IH}\) \(R(n - 1)\), \(B \Rightarrow^* y\). So build the derivation \(S \Rightarrow SbB \Rightarrow bB\) (since \(S\) is nullable) \(\Rightarrow^* by = x\). Thus \(P(n)\) holds in both subcases that apply.

\((Q)\) \#\(b(x)\) − \#\(a(x)\) = 1 mod 3 Again since \(x \neq \epsilon\), we have two subcases:

(i) \(x\) begins with \(a\). Then \(x = ay\) for some string \(y\) of length \(n - 1\), and necessarily \#\(b(y)\) − \#\(a(y)\) = 2 mod 3, so that subtracting 1 for the \(a\) in front leaves 1 mod 3. Since \(|y| = n - 1 < n\), we may apply \(R(n - 1)\) from \(\text{IH}\) to get \(B \Rightarrow^* y\). Then we build the derivation \(A \Rightarrow ab \Rightarrow^* ay = x\), showing \(Q(n)\) as needed toward \(P'(n)\) in this subcase.

(ii) \(x\) begins with \(b\). Then \(x = by\) where \#\(b(y)\) − \#\(a(y)\) = 0 mod 3. By \(\text{IH}\) \(R(n - 1)\), \(S \Rightarrow^* y\). So build \(A \Rightarrow bS \Rightarrow^* by = x\). Thus \(Q(n)\) and hence \(P'(n)\) holds in both subcases that apply.

\((R)\) \#\(b(x)\) − \#\(a(x)\) = 2 mod 3. Here owing to the shape of the grammar we use a different case breakdown:

(i) For some prefix \(u\) of \(x\), \#\(b(u)\) − \#\(a(u)\) = 1 mod 3. Take \(v\) to be the leftover part of \(x\), i.e., so that \(x = uv\). Then \#\(b(v)\) − \#\(a(v)\) = 1 mod 3 too. Since neither \(u\) nor \(v\) can be empty, both of them have length strictly less than \(n\). Hence \(\text{IH}\) \(Q(|u|)\) and \(Q(|v|)\) combine to give \(A \Rightarrow^* u\) and \(A \Rightarrow^* v\). Then \(B \Rightarrow AA \Rightarrow^* uA \Rightarrow^* uv = x\), which is what we needed to show \(R(n)\) and hence \(P'(n)\) in this case. (ii) The negation of (i), i.e. there is no place in \(x\) at which the current difference is 1 mod 3. The only way this can happen is if the difference \#\(b(u)\) − \#\(a(u)\) stays at 0 for awhile, then goes to −1 necessarily following an \(a\). Then \(x = uav\) where we also must have \#\(b(v)\) − \#\(a(v)\) = 0. By \(\text{IH}\) \(P(|u|)\) and \(P(|v|)\), since \(|u|\) and \(|v|\) are each at most \(n - 1\), we have \(S \Rightarrow^* u\) and \(S \Rightarrow^* v\). Then we build \(B \Rightarrow SaS \Rightarrow^* uSaS \Rightarrow^* uav = x\), thus showing \(R(n)\) in both subcases.

Thus \(P'(n)\) holds in all (sub)cases, and the induction goes through. In the proof, we did not use the rules \(S \Rightarrow AB\) and \(A \Rightarrow aa\), so they can be deleted without changing the language. (Note that \(A \Rightarrow aa\) can be simulated directly by \(A \Rightarrow aB \Rightarrow aSaS \Rightarrow aaS \Rightarrow aa\) since \(S\) is nullable, but the rule \(S \Rightarrow AB\) does not have a direct simulation, because the other non-epsilon rules for \(S\) all introduce terminals.)