4. **Problem:** Prove that the class of context-free languages is closed under unions, concatenations, and the Kleene star operation, i.e. if the languages $L_1, L_2 \subseteq \Sigma^*$ are context-free, then so are the languages $L_1 \cup L_2, L_1 L_2$ and $L_1^*$. 

**Solution:** Let $L_1$ and $L_2$ be context-free languages that are defined by grammars $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$. In addition we require that $(V_1 - \Sigma_1) \cap (V_2 - \Sigma_2) = \emptyset$. That is, the grammars may not have any common nonterminals. Since the nonterminals may be renamed if necessary, this is not an essential limitation. 

**Union:** Let $S$ be a new nonterminal and $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \rightarrow S_1 | S_2\}, S)$. Now $L(G) = L(G_1) \cup L(G_2) = L_1 \cup L_2$. This holds, since the initial symbol $S$ may derive only $S_1$ or $S_2$, and they in turn may derive only strings that belong to the respective languages. (If the sets of nonterminals were not disjoint, this would not hold). 

**Concatenation:** The language $L_1 L_2$ is defined by the following grammar: $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}, S)$ 

**Kleene star:** The language $L_1^*$ is defined by the following grammar: $G = (V_1 \cup \{S\}, \Sigma_1, R_1 \cup \{S \rightarrow \varepsilon | SS_1\}, S)$ 

5. **Problem:** Design a context-free grammar describing the syntax of simple “programs” of the following form: a program consists of nested for loops, compound statements enclosed by `begin-end` pairs and elementary operations a. Thus, a “program” in this language looks something like this:

```plaintext
a;
for 3 times do
begin
   for 5 times do a;
   a; a
end.
```

For simplicity, you may assume that the loop counters are always integer constants in the range 0, . . . , 9.

**Solution:** The context-free grammars of programming languages are most often defined so that the alphabet consists of all syntactic elements (lexemes) that occur in the language. In this case numbers, a, and reserved words are lexemes. We divide the parsing of a program into two parts:

(a) The program text is transformed into a string of lexemes using a finite state automaton;

(b) The parse tree of the lexeme string is constructed.

The given grammar can be formalized in many ways, this is one possible interpretation:

$$G = (V, \Sigma, P, C)$$
$$V = \{C, S, N, \textbf{begin, do, end, for, times, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, }, a\}$$
$$\Sigma = \{\textbf{begin, do, end, for, times, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, }, a\}$$
Here the nonterminal \( S \) denotes a statement, \( C \) a compound statement, and \( N \) a number. The rules of the grammar are defined as follows:

\[
P = \{ C \rightarrow S \mid S;C \\
S \rightarrow a \mid \text{begin } C \text{ end } \mid \text{for } N \text{ times } do \ S \\
N \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \}
\]

For example, the program in the problem text has the following parse tree:

6. **Problem**: Prove that the following context-free grammar is ambiguous:

\[
S \rightarrow \text{if } b \text{ then } S \\
S \rightarrow \text{if } b \text{ then } S \text{ else } S \\
S \rightarrow \text{s.}
\]

Design an unambiguous grammar that is equivalent to the grammar, i.e. one that generates the same language.

**Solution**: A context-free grammar is ambiguous if there exists a word \( w \in L(G) \) such that \( w \) has at least two different parse trees. The simplest word for the given grammar that has this property is:

\[
\text{if } b \text{ then if } b \text{ then } s \text{ else } s.
\]

Its two parse trees are:
Usually we want to associate an else-branch to the closest preceding if-statement. In this case the former tree corresponds to this practice.

We define a grammar $G$ as follows:

$$G = (V, \Sigma, P, S)$$

$$V = \{ S, B, U, s, b, \text{if, then, else} \}$$

$$\Sigma = \{ s, b, \text{if, then, else} \}$$

$$P = \{ S \rightarrow B \mid U \}$$

$$B \rightarrow \text{if } b \text{ then } B \text{ else } B \mid s$$

$$U \rightarrow \text{if } b \text{ then } S \mid \text{if } b \text{ then } B \text{ else } U \}$$

Here the nonterminal $B$ is used to derive balanced programs where each if-statement has both then- and else-branches. The nonterminal $U$ derives those if-statements that do not have an else-branch.