T-79.1001/2 **Syksy 2005**

Introduction to Theoretical Computer Science (T/Y)

Session 5

Answers to demonstration exercises

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 $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}, S\}$. 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:$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \to S_1 S_2\}, S\}$

   Kleene star: The language L_1^* is defined by the following grammar: $G = (V_1 \cup \{S\}, \Sigma_1, R_1 \cup \{S \to \epsilon | S S_1\}, S\}$

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:

```
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.

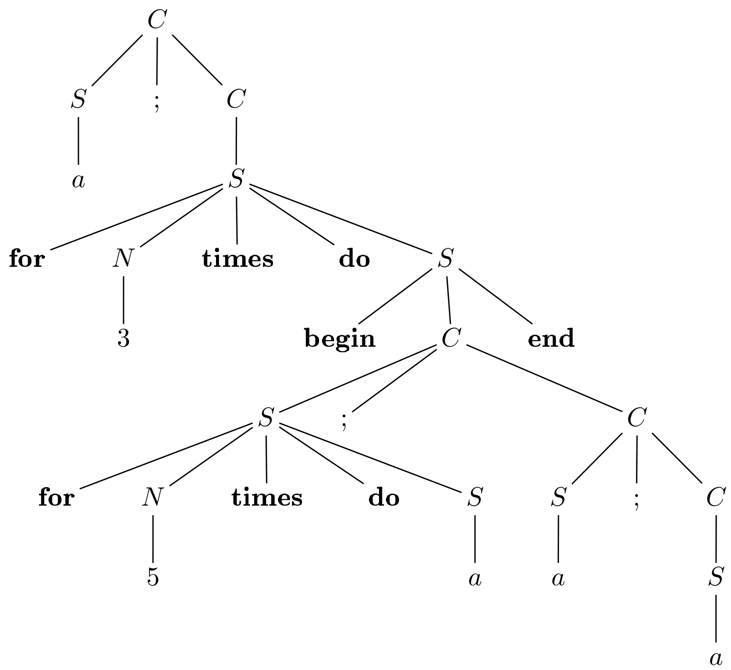
   The given grammar can be formalized in many ways, this is one possible interpretation:

$$
\begin{aligned}
G =& (V, \Sigma, P, C) \\
V =& \{C, S, N, \mathbf{begin}, \mathbf{do}, \mathbf{end}, \mathbf{for}, \mathbf{times}, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ;, a\} \\
\Sigma =& \{\mathbf{begin}, \mathbf{do}, \mathbf{end}, \mathbf{for}, \mathbf{times}, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ;, a\}
\end{aligned}
$$

Here the nonterminal S denotes a statement, C a compound statement, and N a number. The rules of the grammar are defined as follows:

$$\begin{aligned} P = \{ & C \rightarrow S \mid S; C \\ & S \rightarrow a \mid \textbf{begin } C \textbf{ end} \mid \textbf{for } N \textbf{ times do } S \\ & N \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \} \end{aligned}$$

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



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

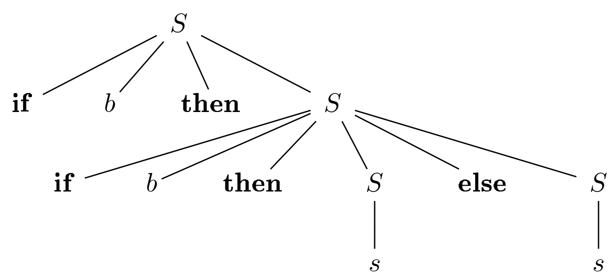
$$\begin{aligned} S &\rightarrow \textbf{if } b \textbf{ then } S \\ S &\rightarrow \textbf{if } b \textbf{ then } S \textbf{ else } S \\ S &\rightarrow s. \end{aligned}$$

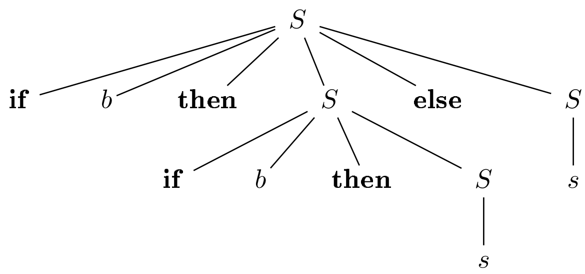
Design an unambiguous grammar that is equivalent to the grammar, i.e. one that generates the same language.

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:

$$\textbf{if } b \textbf{ then if } b \textbf{ then } s \textbf{ else } s.$$

Its two parse trees are:





Usually we want to associate an **else**-branch to the closest preceeding **if**-statement. In this case the former tree corresponds to this practice.

We define a grammar G as follows:

$$
\begin{aligned}
G &= (V, \Sigma, P, S)\\
V &= \{S, B, U, s, b, \textbf{if}, \textbf{then}, \textbf{else}\}\\
\Sigma &= \{s, b, \textbf{if}, \textbf{then}, \textbf{else}\}\\
P &= \{S \rightarrow B \mid U\\
&\qquad B \rightarrow \textbf{if}\ b\ \textbf{then}\ B\ \textbf{else}\ B\ \mid s\\
&\qquad U \rightarrow \textbf{if}\ b\ \textbf{then}\ S \mid \textbf{if}\ b\ \textbf{then}\ B\ \textbf{else}\ U\}
\end{aligned}
$$

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.