T-79.1001/2 **Syksy 2007**
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_1L_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 $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \rightarrow S_1 \mid S_2\}, S\}$. 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_1L_2 is defined by the following grammar:$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \rightarrow S_1S_2\}, S\}$

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

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:

```
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.

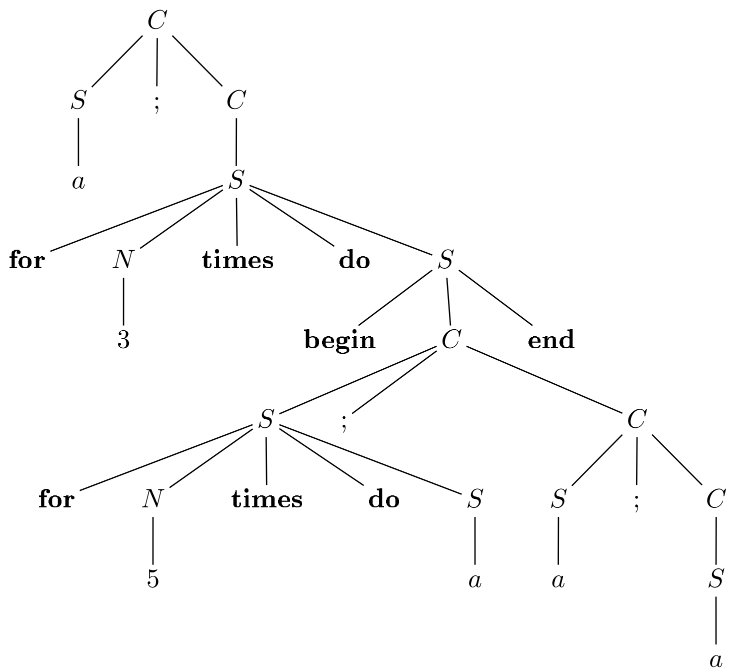
   The given grammar can be formalized in many ways, this is one possible interpretation:

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

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

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

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



6. **Problem:** In the modern WWW-page description language XML designers can construct their own "data type definitions" (abbr. DTD), which are essentially context-free grammars describing the structure of the text or other data displayed on the page. Acquiant yourself with the notation used in this XML/DTD description language and give a context-free grammar corresponding to the following XML/DTD description:

```
<!DOCTYPE Book [
   <!ELEMENT Book (Title, Chapter+)>
   <!ATTLIST Book Author CDATA #REQUIRED>
   <!ELEMENT Title (#PCDATA)>
   <!ELEMENT Chapter (#PCDATA)>
   <!ATTLIST Chapter id ID #REQUIRED>
]>
```

Solution:

The DTD-description defines the structure for a book. There are two kinds of things in the definition: *elements* and *attributes*. The idea is that the book itself consists of the elements and attributes add some meta-information to the elements.

In general, it is not possible to express the semantics of the attributes using only context-free grammars and we need stronger *attribute grammars* for them. However, we can capture the attribute syntax with the grammar.

First, we consider the only the structure the elements. The first element definition

```
<!ELEMENT Book (Title, Chapter+)>
```

tells us that a book contains a title and a sequence of chapters. The '+'-sign tells us that there has to be at least one chapter. The next line:

```
<!ELEMENT Title (#PCDATA)>
```

tells us that a title is a sequence of character data. We will abstract the data away here, and define an alphabet symbol **data** to denote any possible data string. In a real implementation we would use a lexer to identify the data blocks so that the parser of the grammar could work on the abstracted level.

Finally, the line:

```
<!ELEMENT Chapter (#PCDATA)>
```

tells us that a chapter is again character data.

With these definitions we can define the book structure with the following grammar:[1]

$$\begin{aligned} \mathit{Book} &\rightarrow \mathit{Title}\ \mathit{Chapters} \\ \mathit{Title} &\rightarrow \mathbf{data} \\ \mathit{Chapters} &\rightarrow \mathit{Chapter}\ \mathit{Chapters} \mid \mathit{Chapter} \\ \mathit{Chapter} &\rightarrow \mathbf{data} \end{aligned}$$

Now we extend this grammar to coincide with the XML syntax. A syntactic element A starts with an opening tag `<A>` and ends with the corresponding closing tag `</A>`. When we add these to the grammar, we get:

$$\begin{aligned} \mathit{Book} &\rightarrow \langle \mathbf{Book} \rangle\ \mathit{Title}\ \mathit{Chapters}\ \langle/\mathbf{Book}\rangle \\ \mathit{Title} &\rightarrow \langle \mathbf{Title} \rangle\ \mathbf{data}\ \langle/\mathbf{Title}\rangle \\ \mathit{Chapters} &\rightarrow \mathit{Chapter}\ \mathit{Chapters} \mid \mathit{Chapter} \\ \mathit{Chapter} &\rightarrow \langle \mathbf{Chapter} \rangle\ \mathbf{data}\ \langle/\mathbf{Chapter}\rangle \end{aligned}$$

The syntax for attributes in XML is that we add them inside the opening tag. An attribute consists of a name-value pair **name** $=$ **value**:

$$\begin{aligned} \mathit{Book} &\rightarrow \langle\ \mathbf{Book}\ \mathit{BookAttributes}\ \rangle\ \mathit{Title}\ \mathit{Chapters}\ \langle/\mathbf{Book}\rangle \\ \mathit{Title} &\rightarrow \langle \mathbf{Title} \rangle\ \mathbf{data}\ \langle/\mathbf{Title}\rangle \\ \mathit{Chapters} &\rightarrow \mathit{Chapter}\ \mathit{Chapters} \mid \mathit{Chapter} \\ \mathit{Chapter} &\rightarrow \langle\ \mathbf{Chapter}\ \mathit{ChapterAttributes}\ \rangle\ \mathbf{data}\ \langle/\mathbf{Chapter}\rangle \\ \mathit{BookAttributes} &\rightarrow \mathbf{author} = \mathbf{data} \\ \mathit{ChapterAttributes} &\rightarrow \mathbf{id} = \mathbf{data} \end{aligned}$$

[1] The symbols written with *italics* are non-terminals while those in **bold** are terminals.