Solutions to Supplementary Problems

S5.1 Prove that the language $L = \{w \mid w \text{ contains equally many } a\text{'s as } b\text{'s}\}$ is not regular, and design a context-free grammar generating it.

Solution. We shall use the following fundamental property of regular languages to establish the result:

Pumping Lemma: *If L is a regular language, then there exists an integer $n \geq 1$ such that every string $x \in L$ with $|x| \geq n$ can be divided in three parts, $x = uvw$, so that: (1) $|uv| \leq n$, (2) $|v| \geq 1$, and (3) $uv^k w \in L$ for every $k \geq 0$.*

Suppose (for a contradiction) that L is regular. Let n be the threshold string length provided by the Pumping Lemma in this case, and consider string $x = a^n b^n \in L$. Now since $|x| = 2n \geq n$, it should be possible to divide x into three parts u, v, and w so that all three conditions of the lemma hold. We shall argue that such a partition is in fact not possible, which contradicts the assumption that L would be regular.

Assume, to the contrary, that a partioning of the given x as $x = uvw$ satisfying conditions (1)–(3) is possible. Then it follows from (1) that for some $i, j \geq 0$, $i + j \leq n$:

$$u = a^i, \quad v = a^j, \quad w = a^{n-(i+j)} b^n \,.$$

From (2) we know that $j \geq 1$. For these values of i and j, also condition (3) should hold for any $k \geq 0$. So let us try in particular $k = 0$:

$$uv^0 w = uw = a^i a^{n-(i+j)} b^n = a^{n-j} b^n \notin L,$$

which contradicts condition (3).[1] Hence the Pumping Lemma does not apply to language L, and L is not regular.

A grammar G generating the given language L of "ab-balanced strings" is:

$$S \to aSbS \mid bSaS \mid \varepsilon$$

The first rule of the grammar expresses the condition: "If the string starts with an a, then at some position in the string there must be a matching b. The substrings between these a and b and following the b may be any balanced strings". The second rule expresses the corresponding condition for strings starting with a b.

S5.2 Design a context-free grammar that describes simple programs consisting of nested `for` loops, compound statements enclosed by `begin`-`end` pairs and elementary operations `a`. An example "program" in this language is:

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

[1] If you find this kind of "0-pumping" counterintuitive, one could equally well reach the conclusion by choosing $k = 2$, i.e. "pumping" the v-segment twice: $uv^2 w = a^i a^{2j} a^{n-(i+j)} b^n = a^{n+j} b^n \notin L$; contradiction.

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

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 the numerals 0,1,...,9, the "atomic operation" symbol a, and the reserved words are lexemes.

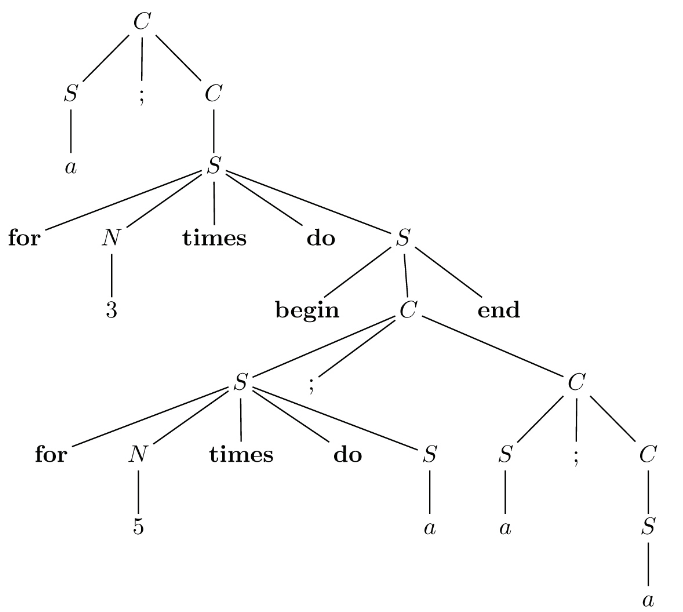
A grammar for the given target language can be designed in many ways. This is one possible approach:

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

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

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

For example, the program in the problem text has the following parse tree (an alternative way to describe derivations in the grammar):



S5.3 In the data description language XML, designers can construct their own "data type definitions" (abbr. DTD), which are essentially context free grammars describing the structure of the data contained in a file. Acquaint yourself with the notation used in this XML/DTD description language (from e.g. `http://www.rpbourret.com/xml/xmldtd.htm`), 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 a stronger formalism such as *attribute grammars* for them. However, a basic context-free grammar can be used to describe the attribute syntax.

First, we consider 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 extract from the DSD code the following simple grammar that describes the structure of a book:[2]

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

If one wishes, this grammar can be extend 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

[2] The symbols written with *italics* are variables while those in **bold** are terminals.

add these to the grammar, we get:

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

The syntax for attributes in XML involves adding them inside the opening tag. An attribute consists of a name-value pair **name = value**:

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