T-79.148 **Spring 2002**
Introduction to Theoretical Computer Science
Tutorial 6, 5–7 March
Problems

Homework problems:

1. Design a context-free grammar that characterises, at an appropriately general level, the structure of a typical newspaper article: headline, caption, text body, subheadings etc.

2. A *palindrome* is a string w such that $w = w^R$, e.g. “MADAMIMADAM”, “ABLEWASIEREISAWELBA”. Consider the set of palindromes over the alphabet $\{a, b\}$:
$$\mathrm{PAL} = \{w \in \{a, b\}^* \mid w = w^R\}.$$
   (a) Prove that this language is not regular.
   (b) Design a context-free grammar generating the language.

3. The languages generated by the following context-free grammars are all regular. Give for each language a regular expression describing it.
   (a) $\{S \to AS \mid \varepsilon, \quad A \to a \mid b\}$
   (b) $\{S \to SSS \mid a \mid b\}$
   (c) $\{S \to AB, \quad A \to aAa \mid bAb \mid \varepsilon, \quad B \to aB \mid bB \mid \varepsilon\}$

Demonstration problems:

4. *Pattern expressions* are a generalisation of regular expression used e.g. in some text editing tools of UN*X-type operating systems. In addition to the usual regular expression constructs, a pattern expression may contain string variables, inducing the constraint that any two appearances of the same variable must correspond to the same substring. Thus e.g. aXb^*Xa and $aX(a\cup b)^*YX(a\cup b)^*Ya$ are pattern expressions over the alphabet $\{a, b\}$. The first one of these describes the language $\{awb^nwa \mid w \in \{a, b\}^*,\ n \geq 0\}$. Prove that pattern expressions are a proper generalisation of regular expressions, i.e. that pattern expressions can be used to describe also some nonregular languages.

5. Prove that the language $\{w \in \{a, b\}^* \mid w \text{ contains equally many } a\text{'s and } b\text{'s}\}$ is not regular, and design a context-free grammar generating it.

6. 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.