Software-Modellierung und Verifikation
Informatik 2
Prof. J.-P. Katoen
RWTH Aachen

Priv.-Doz. T. Noll noll@cs.rwth-aachen.de
Ch. Jansen christina.jansen@cs.rwth-aachen.de

8. Exercise sheet *Compiler Construction 2010*

Due to Wed., 22th December 2010, *before* the exercise course begins.
Hand in your solutions in groups of three or four!

Exercise 8.1: **(4 points)**

Consider the language $L = \{a^{2^n} | n \in \mathfrak{n}\}$. Notice that L is not context-free! Provide an attributed grammar for the language given by $\{a\}^*$. This grammar should determine – via a synthesized boolean attribute at the starting symbol – whether a derived word is contained in L. For this purpose do only use plain functions like addition, multiplication or equality testing, but not testing if a function is a power of two etc.
Generate the corresponding equality system for the input word a^4 and solve it.

Exercise 8.2: **(3+1 points)**

Consider the following grammar:

$$\begin{array}{lllll} S & \rightarrow & A & i_1.1 = 1 \\ & & & i_2.1 = 2 \\ & & & s_1.0 = s_2.1 \end{array} \qquad \begin{array}{lllll} A & \rightarrow & Aa & i_1.1 = s_1.1 \\ & & & i_2.1 = i_1.0 \\ & & & s_1.0 = s_2.1 \\ & & & s_2.0 = 0 \end{array}$$

$$\begin{array}{lllll} A & \rightarrow & a & s_1.0 = 0 \\ & & & s_2.0 = i_1.0 \end{array} \qquad \begin{array}{lllll} A & \rightarrow & b & s_1.0 = i_2.0 \\ & & & s_2.0 = 0 \end{array}$$

a) Show that the grammar is not circular using the method from the lecture.

b) Modify the attribution in a way that it violates the non-circularity of the grammar.

Exercise 8.3: **(4 points)**

Consider the following grammar for a simple programming language:

Program	$\rightarrow$	**var** *Decl* ; *Statement*
Decl	$\rightarrow$	**id** , *Decl* | **id**
Statement	$\rightarrow$	**id** := *AExpr* | **begin** *StmList* **end** | **if** *BExpr* **then** *Statement* | **while** *BExpr* **do** *Statement*
StmList	$\rightarrow$	*Statement* ; *StmList* | *Statement*
AExpr	$\rightarrow$	**id** | **number** | (*AExpr* **arop** *AExpr*)
BExpr	$\rightarrow$	*AExpr* **relop** *AExpr*

Extend the grammar by providing attributes generating a symbol table containing every declared variable. Test whether variables are declared multiple times! Start by allocating to every terminal symbol **id** its variable name by means of synthesized attributes. The symbol table should contain at least the address of the variable. For simplicity you may assume, that every variable needs one storing unit, i.e. n variables obtain addresses 1 to n.
Your attributed grammar should test whether all variable names occuring in a program are declared. Furthermore it should collect the addresses of all variables occuring in the statement part of the program in a synthesized attribute of *Program* , which maintains their order of appearance.