



INF 5110: Compiler construction

Spring 2023

Series 2

9. 2. 2023

Topic: Context free grammars

Issued: 9. 2. 2023

This exercise set covers more than one lecture. It's about grammars, and partly for the lectures about *parsing*. We might not be able to cover it within 2 hours.

Exercise 1 (First- and follow sets) Compute the *First* and *Follow*-sets for the grammar Figure 1.

$$\begin{aligned} \text{exp} &\rightarrow \text{term exp}' \\ \text{exp}' &\rightarrow \text{addop term exp}' \mid \epsilon \\ \text{addop} &\rightarrow + \mid - \\ \text{term} &\rightarrow \text{factor term}' \\ \text{term}' &\rightarrow \text{mulop factor term}' \mid \epsilon \\ \text{mulop} &\rightarrow * \\ \text{factor} &\rightarrow (\text{exp}) \mid \mathbf{number} \end{aligned}$$

Figure 1: Expression grammar (left-recursion removed)

Exercise 2 (Nullable) Describe an algorithm that finds all nullable non-terminals without first finding the first-sets.

Exercise 3 (Associativity and precedence) Take the binary ops $+$, $-$, $*$, $/$ and \uparrow . Let's agree also on the following precedences and associativity

op	precedence	associativity
$+$, $-$	low	left assoc.
$*$, $/$	higher	left. assoc.
\uparrow	highest	right. assoc

Write an *unambiguous* grammar that captures the given precedences and associativities (of course, directly with a BNF grammar, without allowing yourself specifying those requirements as extra side-conditions).

Exercise 4 (Tiny grammar) For the grammar given answer the following questions:

- Is the grammar *unambiguous*?

- How can we change the grammar, so that TINY allows empty statements?
- How can we arrange it that semicolons are required *in between* statements, not *after* statements?
- What's the precedence and associativity of the different operators?

```

program  →  stmts
stmts    →  stmts ; stmt | stmt
stmt     →  if-stmt | repeat-stmt | assign-stmt
         |  read-stmt | write-stmt
if-stmt  →  if expr then stmt end
         |  if expr then stmt else stmt end
repeat-stmt → repeat stmts until expr
assign-stmt → identifier := expr
read-stmt  → read identifier
write-stmt → write expr
expr       → simple-expr comparison-op simple-expr | simple-expr
comparison-op → < | =
simple-expr  → simple-expr addop term | term
addop       → + | -
term        → term mulop factor | factor
mulop       → * | /
factor      → ( expr ) | number | identifier

```

References