Universitetet i Oslo
Institutt for Informatikk
PMA, PSE
Axelsen, Krogdahl, Møller-Pedersen, Steffen

INF 5110: Compiler construction

Spring 2017

Series 3

8. 2. 2017

Topic: Chapter 4: grammars

Issued: 8. 2. 2017

Exercise 1 (LL(1)) Check if the following grammar is LL(1)?

$$S \rightarrow (S)S \mid \epsilon$$

Exercise 2 (Ambiguity) Given the following grammar.

$$\begin{array}{lcl} exp & \rightarrow & exp + exp \mid (\, exp\,) \mid \textbf{if}\ exp\ \textbf{then}\ exp\ \textbf{else}\ exp \mid var \\ var & \rightarrow & \ldots \end{array}$$

1. Come up with an *unambiguous* grammar for the language of the given grammar, where
   (a) addition is left-associative, and where
   (b) **if** x **then** y **else** $z + y$ is meant to mean **if** x **then** y **else** $(z + y)$.
2. Why don't we have a dangling else problem here?

Exercise 3 (Ambiguity) Given the following grammar.

$$\begin{array}{lcl} exp & \rightarrow & exp\ op\ exp \mid (\, exp\,) \mid \textbf{number} \\ op & \rightarrow & + \mid - \mid * \mid / \mid \uparrow \mid < \mid = \end{array}$$

Do the following things.[1]

1. The grammar is pretty ambiguous. Make an unambiguous grammar capturing the same language, under the following side conditions

	precedence	assoc
$\uparrow$	highest (3)	right
$*, /$	level 2	left
$+, -$	level 1	left
$<, =$	0	non-associative

2. Give recursive-descent procedures for each non-terminal to check the grammar (using also loops, if advisable). Divide the terminals representing *op* in an appropriate manner

[1] There's a certain amount of repetition here, we won't go through everything during class-time, but a proposal for solution will be available.

3. Based on the previous point: add tree-building code into the procedures in such a way that sequences of exponentiations $\uparrow$ are treated appropriately in the sense that the tree reflects the intended right-associativity.

4. Take the unambiguous grammar done in the first point, remove left-recursion, and do left-factorization (without destroying unambiguity).

5. Check whether the resulting grammar is LL(1).

References

[1] K. Louden. *Compiler Construction, Principles and Practice.* PWS Publishing, 1997.