Dr. Annette Bieniusa
M.Sc. Peter Zeller

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

CLP: LR-parsing (WS 2018)

This exercise sheet will be discussed in the exercise sessions on October 31.

Exercise 1 Ambiguous grammars

a) Consider the following grammar (given in Java CUP Syntax):

```
expr ::=
       expr LESS expr
     | expr AND expr
     | NUMBER
     | ID
     ;
```

Show that this grammar is ambiguous.

b) To fix the ambiguity in the grammar your colleague suggests the following grammar:

```
expr ::=
       atomicExpr operator expr
     | atomicExpr
     ;

atomicExpr ::=
       NUMBER
     | ID
     ;

operator ::=
       LESS
     | AND
     ;
```

What is the problem with this solution? Can you give a better solution for the problem?

Exercise 2 Towards LR(0)-DFA construction

Let Γ be a grammar with start symbol S and productions:

$$\begin{array}{lcl} S & \rightarrow & T\ V \\ T & \rightarrow & i \mid (\ T\ x\ T\ ) \\ V & \rightarrow & i \mid n \mid (\ V\ i\ V\ ) \end{array}$$

a) Give an execution of an LR parser accepting the input `(i x i) ((i i n) i i)`.
(Complete the table below the parser state for each step)

Stack	Input	Next Action
	(i x i) ((i i n) i i)	shift
(	i x i) ((i i n) i i)	
⋮	⋮	⋮

b) How did you decide wether to reduce using $T \rightarrow i$ or $V \rightarrow i$?

c) How did you decide wether to shift i or reduce using $V \rightarrow i$?

d) Describe how one can pick the next action by just looking at the current stack.

e) Give a regular expression describing the state of the stack, such that the regular expression matches the stack iff the next action should be "reduce using $V \rightarrow i$".

Exercise 3 LR(0)-DFA construction

a) Construct the LR(0)-DFA for the grammar from exercise 2.

b) Construct the LR(0)-DFA for the following grammar (given in Java CUP Syntax).

```
// this first rule would not be written down in CUP
Start ::= statements EOF;

statements ::=
        stmt statements
      | /* empty */
      ;

stmt ::=
        type ID SEMI
      | expr SEMI
      | expr EQ expr SEMI
      ;

type ::= ID;

expr ::=
        ID
      | expr DOT ID
      ;
```