apl. Prof. Dr. T. Noll, Prof. Dr. U. Naumann, F. Gretz, C. Jansen, M. Förster, J. Lotz

First Name: ____________________

Second Name: ____________________

Matriculation Number: ____________________

Degree Programme (please mark):

- ○ CS Bachelor
- ○ CS Master
- ○ CS Lehramt
- ○ SSE Master
- ○ Other: ____________________

	Σ Points	**Points obtained**
Exercise 1	**5**	
Exercise 2	**10**	
Exercise 3	**10**	
Exercise 4	**10**	
Exercise 5	**10**	
Exercise 6	**15**	
Σ	**60**	

- Mark every sheet with your **matriculation number**.
- Check that your copy of the exam consists of **8 sheets**.
- Duration of exam: **120 minutes**.
- No helping materials (e.g. books, notes, slides) are permitted.
- Give your solution on the respective sheet. Also use the backside if necessary. If you need more paper, ask the assistants.
- Write with blue or black ink; do **not** use a pencil or red ink.
- Make sure all electronic devices are switched off and are nowhere near you.
- Any attempt at deception leads to failure for this exam, even if it is detected only later.

Question 1 (Multiple Choice): (5 Points)

T(rue) or F(alse)? Please check! (Each correct answer gives 0.5 points, wrong answers are ignored.)

1. Lexical analysis:

◯T ◯F Prefix-free languages can be analysed with a scanner using the first-match principle only (that is, without considering the longest match). (Reminder: a language is called prefix free if it does not contain a proper prefix of any of its elements.)

◯T ◯F The worst-case time complexity of the backtracking automaton (as presented in the lecture) is quadratic in the length of the input word.

◯T ◯F Every input word that can be decomposed with respect to the given regular expressions also has a first-longest-match decomposition.

2. Syntax analysis:

◯T ◯F Every grammar that generates exactly one word has the $LL(0)$ property.

◯T ◯F A context-free grammar has the $LL(1)$ property if the lookahead sets of all productions with the same left-hand side are pairwise inequal.

◯T ◯F A context-free grammar that generates an inherently ambiguous language can never be transformed to an equivalent $LR(k)$ grammar (for any k).

3. Semantic analysis:

◯T ◯F An attribute grammar is circular if the dependency graph of each of its syntax trees contains a cycle.

◯T ◯F The language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ can be recognized by an attribute grammar using only synthesized attributes.

4. Code generation:

◯T ◯F In the procedure stack, static links to the I/O frame can only originate in the MAIN frame.

◯T ◯F The following procedure stack could result from the execution of a translated EPL program:

$$13 : 4 : 9 : 1 : 4 : 3 : 2 : 2 : 4 : 4 : 3 : 15 : 1 : 3 : 2 : 12 : 0 : 0 : 0 : 17 : 3$$

Question 2 (Lexical Analysis): (10 Points)

a) Given the regular expressions $\alpha_1 = a^+b$ and $\alpha_2 = (a^+ba)^+$. Perform an FLM-analysis, i.e.

i) Construct the corresponding product automaton A recognising $L(A) = [\![\alpha_1]\!] \cup [\![\alpha_2]\!]$. (Here you can start with the DFAs of the two expressions)

ii) Partition the set of final states to follow the first-match-principle.

iii) Provide a run of the backtracking automaton on the input $w = abaa$.

b) Implement the backtracking automaton for lexical analysis. To this aim fill in the gaps in the skeleton below with java-like pseudo-code. Output the resulting analysis as well as possible errors when reasonable.

You may utilise the following functions:

- LinkedList:
  - isEmpty() returns true if the list is empty, otherwise false.
  - toString() returns a string containing all elements of the list in the correct order
  - getFirst() returns the first element of the list. If the list is empty, it returns null.
  - removeFirst() removes the first element from the list and returns it. If the list is empty, it returns null.
  - addFirst(Object o) adds *o* to the beginning of the list (defined on lists of objects, too).
  - add(Object o) adds *o* to the end of the list (defined on lists of objects, too).
  - clear() removes all of the elements from the list.
- DFA:
  - getNext(Char x, State s) returns the resulting state when taking an *x*-transition in state *s*.
  - isProductive(State s) returns true if *s* is a productive state, otherwise false.
- StatePartition:
  - getToken(State s) returns the token which corresponds to the final state *s* according to the partition. If *s* is not in the set of final states, it returns null.
- Token:
  - toString() returns a string representing the token

```
public void backtrackAutomaton(LinkedList<Char> input, DFA automaton,
                               StatePartition finalStates, State initialState){

    // set to either null or the current token recognised
    Token mode = null;
    // stores the current state of the automaton
    State curState = initialState;
    // stores symbols for backtracking
    LinkedList<Char> backtrackStore = new LinkedList<Char>();
    // the resulting analysis
    LinkedList<Token> result = new LinkedList<Token>();
```

```
        while (! input.isEmpty()) {


            if (mode == null) {









            }
            if (!mode == null) {











            }
            if( input.isEmpty() &&                                          ) {











            }
        }
        if (input.isEmpty() &&                                              ) {


        }
        else if (input.isEmpty() &&                                         ) {


    }   }
```

Question 3 (Syntactic Analysis): (10 Points)

a) Consider the following grammar G:

$$\begin{array}{lcl} S & \rightarrow & aA \mid aB \\ A & \rightarrow & a \mid aA \\ B & \rightarrow & b \mid bB \end{array}$$

i) Show that $G \notin LL(1)$.

ii) Is $G \in LL(2)$? Prove your answer.

b) Show that the set of all regular languages is a proper subset of the set of $LL(1)$ languages, proceeding as follows:

i) Show that every regular language can be generated by an $LL(1)$ grammar.
(**Hint:** regular languages are recognized by deterministic finite automata.)

ii) Give an example of an $LL(1)$ language that is not regular. (You do not have to prove that your language is not regular.)

Question 4 (Semantic Analysis): (10 Points)

The following grammar generates two sets, each with two words over $\Sigma = \{a\}$. We want to use attributes to decide if the two generated sets have a non-empty intersection. An assistant came up with the following solution: (**Hint:** result, nextSelect and length are synthetic attributes. curSelect is inherent. Furthermore you may assume that nextSelect has been set to left initially by some other rules which are irrelevant for this question!)

$$
\begin{array}{llll}
S & \rightarrow & \{C\}\{C\} & \text{result.0} = (\text{length.2} == \text{length.5}) \\
 & & & \text{curSelect.2} = \begin{cases} \text{right } \textbf{if } \text{length.2} \neq \text{length.5} \wedge \text{nextSelect.2} == \text{left} \wedge \text{nextSelect.5} == \text{left} \\ \text{left } \textbf{if } \text{length.2} \neq \text{length.5} \wedge \text{nextSelect.2} == \text{right} \wedge \text{nextSelect.5} == \text{right} \\ \text{right } \textbf{if } \text{length.2} \neq \text{length.5} \wedge \text{nextSelect.2} == \text{right} \wedge \text{nextSelect.5} == \text{left} \\ \text{left } \textbf{if } \text{length.2} \neq \text{length.5} \wedge \text{nextSelect.2} == \text{left} \wedge \text{nextSelect.5} == \text{right} \end{cases} \\
 & & & \text{curSelect.5} = \begin{cases} \text{right } \textbf{if } \text{length.2} \neq \text{length.5} \wedge \text{nextSelect.2} == \text{right} \wedge \text{nextSelect.5} == \text{left} \\ \text{left } \textbf{if } \text{length.2} \neq \text{length.5} \wedge \text{nextSelect.2} == \text{left} \wedge \text{nextSelect.5} == \text{left} \\ \text{right } \textbf{if } \text{length.2} \neq \text{length.5} \wedge \text{nextSelect.2} == \text{right} \wedge \text{nextSelect.5} == \text{right} \\ \text{left } \textbf{if } \text{length.2} \neq \text{length.5} \wedge \text{nextSelect.2} == \text{left} \wedge \text{nextSelect.5} == \text{right} \end{cases} \\
C & \rightarrow & A, B & \text{length.0} = \begin{cases} \text{length.1 } \textbf{if } \text{curSelect.0} == \text{left} \\ \text{length.3 } \textbf{if } \text{curSelect.0} == \text{right} \end{cases} \\
 & & & \text{nextSelect.0} = \text{curSelect.0} \\
A & \rightarrow & a & \text{length.0} = 1 \\
A & \rightarrow & aA & \text{length.0} = 1 + \text{length.2} \\
B & \rightarrow & a & \text{length.0} = 1 \\
B & \rightarrow & aB & \text{length.0} = 1 + \text{length.2}
\end{array}
$$

a) Apply the circularity test to decide whether the attributed grammar is circular!

b) Keep the underlying context-free grammar but define a new (non-circular) attribute grammar over it such that again $\text{result.0} == \text{true}$ at the root node of the derivation tree if and only if the intersection of the two generated sets is not empty. (**Hint:** Using synthetic attributes only suffices here.)

Question 5 (Code Generation): (10 Points)

a) Let the current state of the abstract machine AM for procedures (without parameters) be given by

$$(l, d, p) := (3,\ 10,\ 9:4:45:3:2:4:3:30:5:10:4:40:1:2:\ldots).$$

We assume that the corresponding code contains the following instructions:

```
  ⋮
3:LOAD (0, 2)% (dif, off)
4:LOAD (1, 1)% (dif, off)
5:ADD
6:LT
8:JFALSE(20)
9:
  ⋮
```

Determine the respective machine state after the execution of the next five instructions.

(**Reminder:** the topmost entry of the runtime stack p is to the left, and each activation block is of the form $sl : dl : ra : loc_1 : \ldots : loc_k$.)

b) Extend the code generation for EPL programs by taking counting loops into account. Do **not** explicitely model for- by while-loops!

- The syntax of EPL is extended by statements of the form

$$\textbf{for}\ I := A_1\ \textbf{to}\ A_2\ \textbf{do}\ C \quad \in Cmd$$

  where $I \in Ide$, $A_1, A_2 \in AExpr$, and $C \in Cmd$.
- The interpretation of this construct is as usual: C is executed as long as the value of I is at most A_2, starting with initial value A_1 and incremented by one at the end of each iteration.
- You can assume that the symbol table, $st \in Tab$, contains an entry of the form

$$st(I) = (\texttt{var}, lev, off)$$

  (that is, I is declared as a variable on level $lev \in Lev$ with offset $off \in Off$).
- Further you can assume that the evaluation of the arithmetic expessions A_1 and A_2 pushes its result to the data stack.

Give a definition of the command translation mapping

$$\mathrm{ct}(\textbf{for}\ I := A_1\ \textbf{to}\ A_2\ \textbf{do}\ C, st, a, l)$$

for a starting code address $a \in PC$ and source code level $l \in Lev$.

Question 6 (Algorithmic Differentiation): (15 Points)

Consider numerical programs whose syntax is defined by a grammar with the following production rules:

$$\begin{array}{lcl} A & \rightarrow & v = E\,; \\ E & \rightarrow & sin\,(\,E\,)\,|\,sub\,(\,E\,,\,E\,)\,|\,v \end{array}$$

The start symbol is A. Terminal symbols represent scalar variables (v, for example, words over all lowercase letters), the unary sinus function (sin, corresponding to string "sin"), and binary floating-point subtraction (sub, corresponding to string "−").

a) Give the $LR(0)$ automaton of this grammar.

b) Extend the context-free grammar to an attribute grammar that generates tangent-linear derivative code. Use the floating-point operations "$*$" (binary scalar multiplication) and "cos" (unary cosinus). Give the result of applying the corresponding single-pass compiler to the following program:

```
y=−(y , sin(−(y , x))) ;
```