

apl. Prof. Dr. Thomas Noll, Dipl.-Inform. Friedrich Gretz, Souymodip Chakraborty, M.Sc.

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	10	
Exercise 2	10	
Exercise 3	10	
Exercise 4	10	
Exercise 5	10	
Exercise 6	10	
Σ	60	

- Mark every sheet with your **matriculation number**.
 - Check that your copy of the exam consists of **10 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):**(10 Points)**

Tick the option that you think is correct. (Each correct answer gives 1 point, wrong answers are ignored.)

1. Consider the regular expressions $\alpha_1 = ab^+$ and $\alpha_2 = ba$ and the string $w = abbba$. There exists a first-longest-match analysis of w with respect to α_1 and α_2 .

☐Correct ☐Wrong

2. Once the Backtracking DFA reaches an unproductive state ($q \notin P$), it must abort as there is no analysis of the given input.

☐Correct ☐Wrong

3. For every $k \in \mathbb{N}$ there is a grammar G for which a non-deterministic top-down automaton $NTA(G)$ exists but there is no deterministic top-down automaton $DTA(G)$ with lookahead bounded by k .

☐Correct ☐Wrong

4. Given $DTA(G)$ and a word $w \in \mathcal{L}(G)$ of length l , $DTA(G)$ finds an analysis of w in at most l steps.

☐Correct ☐Wrong

5. A bottom-up parser (LR -parser) reads its input from right to left.

☐Correct ☐Wrong

6. For every $LR(k)$ grammar G we can find an $LR(1)$ grammar G' , an $SLR(1)$ grammar G'' and an $LALR(1)$ grammar G''' which are all equivalent, i.e. $\mathcal{L}(G) = \mathcal{L}(G') = \mathcal{L}(G'') = \mathcal{L}(G''')$.

☐Correct ☐Wrong

7. In order to decide circularity of an attribute grammar, it suffices to check circularity on one derivation tree in which every production of the grammar has been used at least once.

☐Correct ☐Wrong

8. In the lecture languages were categorised into *programming languages*, *intermediate code* and *machine code*. According to these categories the Jasmin code generated by our compiler is an example of...

☐intermediate code ☐machine code

9. The following procedure stack could result from the execution of a translated EPL program:

3 : 2 : 11 : 3 : 2 : 17 : 4 : 3 : 2 : 13 : 0 : 0 : 0 : 5

☐Correct ☐Wrong

10. Consider the following Jasmin program fragment

```
ldc 0
ldc 0
if_icmpeq LABEL1
ldc 0
goto LABEL2
LABEL1:
ldc 1
LABEL2:
ldc 0
ldc 1
if_icmpeq LABEL3
ldc 0
goto LABEL4
LABEL3:
ldc 1
LABEL4:
imul
ifle LABEL5
ldc 0
goto END
LABEL 5:
ldc 1
END:
```

where `ldc` loads a constant on top of the stack, `if_icmpeq` jumps to the given label if the two integers are equal, `imul` multiplies two integers and `ifle` jumps if the integer is less than or equal to zero.

What value is on top of the stack when the execution reaches the `END` label?

☐ 0 ☐ 1

Question 2 (Lexical Analysis):**(10 Points)**

In this question we consider an XML-like language that describes systems of linear (in)equalities. For example, a word of this language can look like:

```
<system>
  <desc>trapezoid example</desc>
  <constraint>y>=0</constraint>
  <constraint>y<=1</constraint>
  <constraint>4*x+2*y<=5</constraint>
  <constraint>3*x-y+1>y</constraint>
</system>
```

a) Give regular expression over the alphabet $\Sigma = \{a, \dots, z, 0, \dots, 9, \square, <, >, =, +, -, *, /\}$ for the following symbol classes:

- opening tag (A tag with some name consisting of letters only. The name is not restricted to match only the ones in the example above!)
- closing tag
- string (Non-empty sequence of letters and blanks.)
- constraint (Equality or inequality between a linear expression with integer coefficients on the left hand side and an integer or variable name on the right hand side, no blanks.)

b) With the regular expressions from above, answer the following questions and argue your point.

- Does the order on the regular expressions matter for the outcome of the *first longest match* (FLM) analysis?
- Could you simply use the *first match* principle (with a particular order on the regular expressions) instead of FLM for scanning the input?

Question 3 (Top Down Syntax Analysis):**(10 Points)**a) Consider the following grammar G :

$$\begin{aligned}
S &\rightarrow Cmd\ S \mid Cmd \\
Cmd &\rightarrow N \mid P \\
N &\rightarrow [\]\ U\ N \mid [\]\ U \\
P &\rightarrow [\text{prob}]\ U\ P \mid [\text{prob}]\ U \\
U &\rightarrow \text{variable} = \text{value}
\end{aligned}$$

- Show by a formal argument that G is not an $LL(1)$ grammar.
- Transform G to an equivalent $G' \in LL(1)$.
You do not have to prove formally that G' has this property.

b) Remember that a context-free grammar $G = (N, V, P, S)$ is in *Greibach normal form* (GNF), if all production rules in P are of the form:

$$A \rightarrow xA_1A_2 \cdots A_n$$

or

$$S \rightarrow \varepsilon$$

where A is a nonterminal symbol, x is a terminal symbol, $A_1A_2 \dots A_n$ is a (possibly empty) sequence of nonterminal symbols not including the start symbol, S is the start symbol, and ε is the empty word.

Formally prove or disprove that every context-free grammar in GNF is an $LL(k)$ grammar for some $k \in \mathbb{N}$.

Question 4 (Bottom Up Syntax Analysis):**(10 Points)**

Consider the following grammar G :

$$\begin{aligned} S &\rightarrow (\text{ assert } F) \\ F &\rightarrow (\text{ or } F F) \mid (> V V) \\ V &\rightarrow \text{ value } \mid \text{ variable } \end{aligned}$$

- a) Compute the $LR(0)$ sets and transitions between them. Either provide the sets and a transition table or depict the goto-DFA. (*Hint: there are 16 sets.*)
- b) Why is this grammar an $LR(0)$ grammar?
- c) Give a run of the corresponding $DBA(G)$ on the input word $w = (\text{ assert } (> \text{ value variable }))$.

Question 5 (Semantic Analysis):**(10 Points)**

a) Consider the following context-free grammar enriched with semantic rules

$$\begin{array}{ll}
 S \rightarrow A & b.0 = isEqual(x.1, y.1) \\
 A \rightarrow AA & x.0 = x.1 + x.2 \\
 & y.0 = y.1 + y.2 \\
 A \rightarrow \mathbf{a} & x.0 = 1 \\
 & y.0 = 0 \\
 A \rightarrow \mathbf{b} & y.0 = 1 \\
 & x.0 = 0
 \end{array}$$

where *isEqual()* is a function that takes two numbers and returns *true* if they are equal and *false* otherwise.

- Construct an attributed syntax tree for the word "baab".
- Set up the corresponding attribute equation system and solve it.
- What is the context-dependent property of a word that is checked by the semantic rules?

b) Now consider another attribute grammar:

$$\begin{array}{ll}
 S \rightarrow C & s.0 = s.1 \\
 & i.1 = 0 \\
 C \rightarrow AB & s.0 = 17 + s.1 \\
 & i.1 = s.2 \\
 & i.2 = s.1 \\
 A \rightarrow C & i.1 = i.0 \\
 & s.0 = s.1 \\
 A \rightarrow \mathbf{x} & s.0 = 24 \cdot i.0 \\
 B \rightarrow \mathbf{y} & s.0 = 42 + i.0
 \end{array}$$

Check whether the attribute grammar is circular. Therefore,

- depict the dependency graph for every production rule,
- iteratively construct $IS(X)$ for every non-terminal X , and
- either explain why the grammar is non-circular or construct a syntax tree with a cyclic dependency graph.

Question 6 (Code Generation):**(10 Points)**

- a) Translate the following EPL program to abstract machine (AM) code.

```

in/out x;
proc P;
    x := x * 2
proc Q;
    x := x + 1
if x >= 1 then
    P()
else
    Q()

```

Hint: On the last two pages of this exam you can find the translation rules from the lecture for your reference.

You may use symbolic addresses!

- b) Assume the abstract machine is currently in state

$(20, d : 6, 3 : 2 : 28 : 4 : 3 : 2 : 17 : p)$

where d and p indicate the remainder of the data- and procedure-stack which are irrelevant for this question. Give the four next states which result from executing the following AM program fragment.

```

.
.
.
20: LIT (7);
21: MUL;
22: STORE(1,1);
23: RET;
.
.
.

```

Reference

$$\begin{aligned}
& \text{trans}(\text{in/out } l_1, \dots, l_n; K.) := 1 : \text{CALL}(a, 0, \text{size}(K)); \\
& \quad 2 : \text{JMP}(0); \\
& \quad \text{kt}(K, \text{st}_{I/O}, a, 1) \\
& \text{kt}(D \ C, \text{st}, a, l) := \text{dt}(D, \text{update}(D, \text{st}, l), l) \\
& \quad \text{ct}(C, \text{update}(D, \text{st}, l), a, l) \\
& \quad a' : \text{RET}; \\
& \text{dt}(D_C \ D_V \ D_P, \text{st}, l) := \text{dt}(D_P, \text{st}, l) \\
& \quad \text{dt}(\epsilon, \text{st}, l) := \epsilon \\
& \text{dt}(\text{proc } l_1; K_1; \dots; \text{proc } l_n; K_n; , \text{st}, l) := \text{kt}(K_1, \text{st}, a_1, l + 1) \\
& \quad \vdots \\
& \quad \text{kt}(K_n, \text{st}, a_n, l + 1) \\
& \quad \text{where } \text{st}(l_j) = (\text{proc}, a_j, \dots, \dots) \text{ for every } j \in [n] \\
& \text{ct}(l := A, \text{st}, a, l) := \text{at}(A, \text{st}, a, l) \\
& \quad a' : \text{STORE}(l - \text{lev}, \text{off}); \\
& \quad \text{if } \text{st}(l) = (\text{var}, \text{lev}, \text{off}) \\
& \text{ct}(l(), \text{st}, a, l) := a : \text{CALL}(ca, l - \text{lev}, \text{loc}); \\
& \quad \text{if } \text{st}(l) = (\text{proc}, ca, \text{lev}, \text{loc}) \\
& \text{ct}(C_1; C_2, \text{st}, a, l) := \text{ct}(C_1, \text{st}, a, l) \\
& \quad \text{ct}(C_2, \text{st}, a', l) \\
& \text{ct}(\text{if } B \text{ then } C_1 \text{ else } C_2, \text{st}, a, l) := \text{bt}(B, \text{st}, a, l) \\
& \quad a' : \text{JFALSE}(a''); \\
& \quad \text{ct}(C_1, \text{st}, a' + 1, l) \\
& \quad a'' - 1 : \text{JMP}(a'''); \\
& \quad \text{ct}(C_2, \text{st}, a'', l) \\
& \quad a''' : \\
& \text{ct}(\text{while } B \text{ do } C, \text{st}, a, l) := \text{bt}(B, \text{st}, a, l) \\
& \quad a' : \text{JFALSE}(a'' + 1); \\
& \quad \text{ct}(C, \text{st}, a' + 1, l) \\
& \quad a'' : \text{JMP}(a); \\
& \text{bt}(A_1 < A_2, \text{st}, a, l) := \text{at}(A_1, \text{st}, a, l) \\
& \quad \text{at}(A_2, \text{st}, a', l) \\
& \quad a'' : \text{LT}; \\
& \text{bt}(\text{not } B, \text{st}, a, l) := \text{bt}(B, \text{st}, a, l) \\
& \quad a' : \text{NOT}; \\
& \text{bt}(B_1 \text{ and } B_2, \text{st}, a, l) := \text{bt}(B_1, \text{st}, a, l) \\
& \quad \text{bt}(B_2, \text{st}, a', l) \\
& \quad a'' : \text{AND};
\end{aligned}$$

Of course, you may also use OR, GT, LEQ, GEQ, EQ and NEQ which are defined analogously.

$$\begin{aligned}
 \text{at}(z, \text{st}, a, l) &:= a : \text{LIT}(z); \\
 \text{at}(l, \text{st}, a, l) &:= \begin{cases} a : \text{LIT}(z); & \text{if } \text{st}(l) = (\text{const}, z) \\ a : \text{LOAD}(l - \text{lev}, \text{off}); & \text{if } \text{st}(l) = (\text{var}, \text{lev}, \text{off}) \end{cases} \\
 \text{at}(A_1 + A_2, \text{st}, a, l) &:= \text{at}(A_1, \text{st}, a, l) \\
 &\quad \text{at}(A_2, \text{st}, a', l) \\
 &\quad a'' : \text{ADD};
 \end{aligned}$$

Of course, you may also use SUB, MUL and DIV which are defined analogously.