

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

Question 1 (Multiple Choice):**(5 Points)**

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

1. Lexical analysis:

☐T ☐F The construction phase of the DFA method to solve the simple matching problem requires polynomial time.

☐T ☐F If the decomposition of a word w w.r.t. regular patterns $\alpha_1, \dots, \alpha_n$ is unique, then the corresponding analysis is unique, too.

2. Syntax analysis:

☐T ☐F If a word w has exactly one leftmost derivation, then w has exactly one syntax tree.

☐T ☐F If $G \in LR(k)$, then $G \in LL(k+1)$.

3. Semantic analysis:

☐T ☐F The declaredness property of identifiers (in programs of the shape *Program* ::= *Declarations Statements*) can be checked without using inherited attributes.

☐T ☐F The attribute equation system of a syntax tree has either exactly one solution or is unsolvable.

4. Code generation:

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

10 : 3 : 5 : 4 : 3 : 2 : 4 : 3 : 2 : 12 : 0 : 0 : 0 : 4 : 3

☐T ☐F To implement WHILE statements and other looping constructs, a runtime stack is required.

5. Algorithmic Differentiation:

☐T ☐F It is not possible to compute all Jacobian matrix entries with a given tangent-linear implementation of a function.

☐T ☐F The tangent-linear model of a mathematical function computes matrix-vector products with the Jacobian matrix.

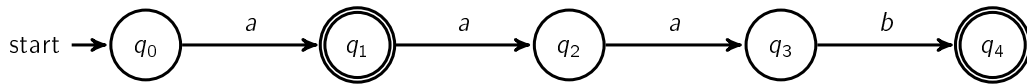
Question 2 (Lexical Analysis):

(4 + 3 + 3 Points)

- a) We consider the backtracking automaton with a bounded backtracking depth d . This bounded backtracking automaton behaves as the original backtracking automaton with the restriction that the length of the lookahead is not allowed to exceed d - instead it backtracks. Please complete the formal definition of the transition function in the backtracking mode:

$$(T, vqaw, W) \vdash \begin{cases} (T_i, q'w, W) & \text{if } \underline{\hspace{4cm}} \\ (T, vaq'w, W) & \text{if } \underline{\hspace{4cm}} \\ (N, q_0vaw, WT) & \text{if } \underline{\hspace{4cm}} \end{cases}$$

Now given the two regular expressions $\alpha_1 = a$ and $\alpha_2 = aaab$. Give a run of the corresponding backtracking automaton with depth $d = 2$ on input word $w = aaaab$ on the following DFA for α_1 and α_2 :



- b) What, in general, is the worst-case runtime of a backtracking automaton with bounded depth d on an input word of length n ? Explain your answer with an example that illustrates this worst-case.

- c) **Note:** this problem is independent of the above!

Let $\text{rev}(L)$ be the language that is obtained from a language L by reversing each word (i.e. reading backwards). Further let $\text{rev}(\alpha)$ be the regular expression that is obtained by reversing α . Formally with $a \in \Omega$, $\alpha, \beta \in RE_\Omega$:

- $\text{rev}(a) = a$
- $\text{rev}(\alpha \circ \beta) = \text{rev}(\beta) \circ \text{rev}(\alpha)$
- $\text{rev}(\alpha^*) = \text{rev}(\alpha)^*$
- $\text{rev}(\alpha + \beta) = \text{rev}(\alpha) + \text{rev}(\beta)$

Given a language L and regular expressions $\alpha_1, \dots, \alpha_n$, prove or disprove the following:

If each word in L has a *first longest match* analysis wrt. $\alpha_1, \dots, \alpha_n$ then each word in $\text{rev}(L)$ has also a *first longest match* analysis wrt. $\text{rev}(\alpha_n), \dots, \text{rev}(\alpha_1)$.

Lösung: _____

a)

$$(T, vqaw, W) \vdash \begin{cases} (T_i, q'w, W) & \text{if } q' \in F^{(i)} \\ (T, vaq'w, W) & \text{if } |va| \leq d \wedge q' \in P \setminus F \\ (N, q_0vaw, WT) & |va| > d \wedge q' \notin F \end{cases}$$

Assuming the states of the automaton are as follows:

$$\rightarrow q_0 \xrightarrow{a} [q_1] \xrightarrow{a} q_2 \xrightarrow{a} q_3 \xrightarrow{b} [q_4]$$

The run would be:

$$\begin{aligned} & (N, q_0aaaab, \epsilon) \\ & \vdash (T_1, q_1aaab, \epsilon) \vdash (T_1, aq_2aab, \epsilon) \vdash (T_1, aaq_3ab, \epsilon) \\ & \vdash (N, q_0aaab, T_1) \vdash (T_1, q_1aab, T_1) \vdash (T_1, aq_2ab, T_1) \\ & \vdash (T_1, aaq_2b, T_1) \vdash (N, q_4, T_1 \cdot T_2) \end{aligned}$$

b) The worst-case runtime is bounded by

$$|w| \cdot d .$$

To see this choose

$$\alpha_1 = a \quad \alpha_2 = a^*b \quad w = a^n \quad d < n .$$

The automaton has to read d symbols before he decides to backtrack and he will do so $(n - d)$ times. Additionally for the last d input symbols it will read the remaining input until the end. The latter number is a constant in d .

c) The claim is wrong. To see this, pick

$$L = \{baa\} \quad \alpha_1 = ba \quad \alpha_2 a^+ .$$

The analysis of (every) word in L is $T_1 \cdot T_2$.

Now, reversing everything we get

$$\text{rev}(L) = \{aab\} \quad \text{rev}(\alpha_1) = ab \quad \text{rev}(\alpha_2) = a^+ .$$

And cannot possibly find an FLM analysis of aab wrt. $\text{rev}(\alpha_2)$ and $\text{rev}(\alpha_1)$.

Question 3 (LL(1)):

(1 + 4 + 3 Points)

a) Consider the following grammar G :

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow aBdAC \mid aA \mid a \\ B &\rightarrow b \\ C &\rightarrow cA \mid dB \end{aligned}$$

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

ii) Transform G such that the result is in $LL(1)$. Argue why!

b) Consider a grammar $G = (N, T, P, S)$.

Show that one direction of the following statement holds and the other one does not.

For all pairs of rules $A \rightarrow \beta, B \rightarrow \gamma \in P$ (where $\beta \neq \gamma$) $la(A \rightarrow \beta) \cap la(B \rightarrow \gamma) = \emptyset \Leftrightarrow G \in LL(1)$.

Lösung:

a) i) To show that $G \notin LL(1)$, consider the following lookahead-sets:

$la(A \rightarrow aBdAC) = \{a\}$, $la(A \rightarrow aA) = \{a\}$ and $la(A \rightarrow a) = \{a\}$. Thus e.g. $la(A \rightarrow aA) = \{a\} \cap la(A \rightarrow a) = \{a\} \neq \emptyset$. And therefore $G \notin LL(1)$.

ii) First apply left factorisation ending up in the following grammar:

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow aA' \\ A' &\rightarrow BdAC \mid A \mid \epsilon \\ B &\rightarrow b \\ C &\rightarrow cA \mid dB \end{aligned}$$

To show that the resulting grammar is in $LL(1)$, calculate the lookahead sets:

$$la(S \rightarrow A) = \{a\}$$

$$la(A \rightarrow aA') = \{a\}$$

$$la(A' \rightarrow BdAC) = \{b\}$$

$$la(A' \rightarrow A) = \{a\}$$

$$la(A' \rightarrow \epsilon) = \{\epsilon, c\}$$

$$la(B \rightarrow b) = \{b\}$$

$$la(C \rightarrow cA) = \{c\}$$

$$la(C \rightarrow dB) = \{d\}$$

Thus the grammar is in $LL(1)$.

b) $\Rightarrow$: Assume the proposition: For all pairs of rules $A \rightarrow \beta, B \rightarrow \gamma \in P$ (where $\beta \neq \gamma$) $la(A \rightarrow \beta) \cap la(B \rightarrow \gamma) = \emptyset$ holds.

Then the proposition is especially true for the choice of $A = B$. With the Characterisation of $LL(1)$ Theorem from the lecture it follows directly that $G \in LL(1)$ holds.

$\Leftarrow$: Disprove by counterexample:

Consider the grammar $G = (\{S, A, B\}, \{a\}, \{S \rightarrow AB, A \rightarrow a, B \rightarrow a\}, S)$.

G is trivially in $LL(1)$ according to the Characterisation Theorem (as there exists exactly one rule per non-terminal), but according to the proposition $la(A \rightarrow a) \cap la(B \rightarrow a) = \emptyset$, $\rightarrow$ Contradiction!

Question 4 (Semantic Analysis):**(8 + 6 Points)****Note:** the following two problems can be solved independently.**a)** Consider the following attributed grammar with an inherited attribute i and a synthesized attribute s :

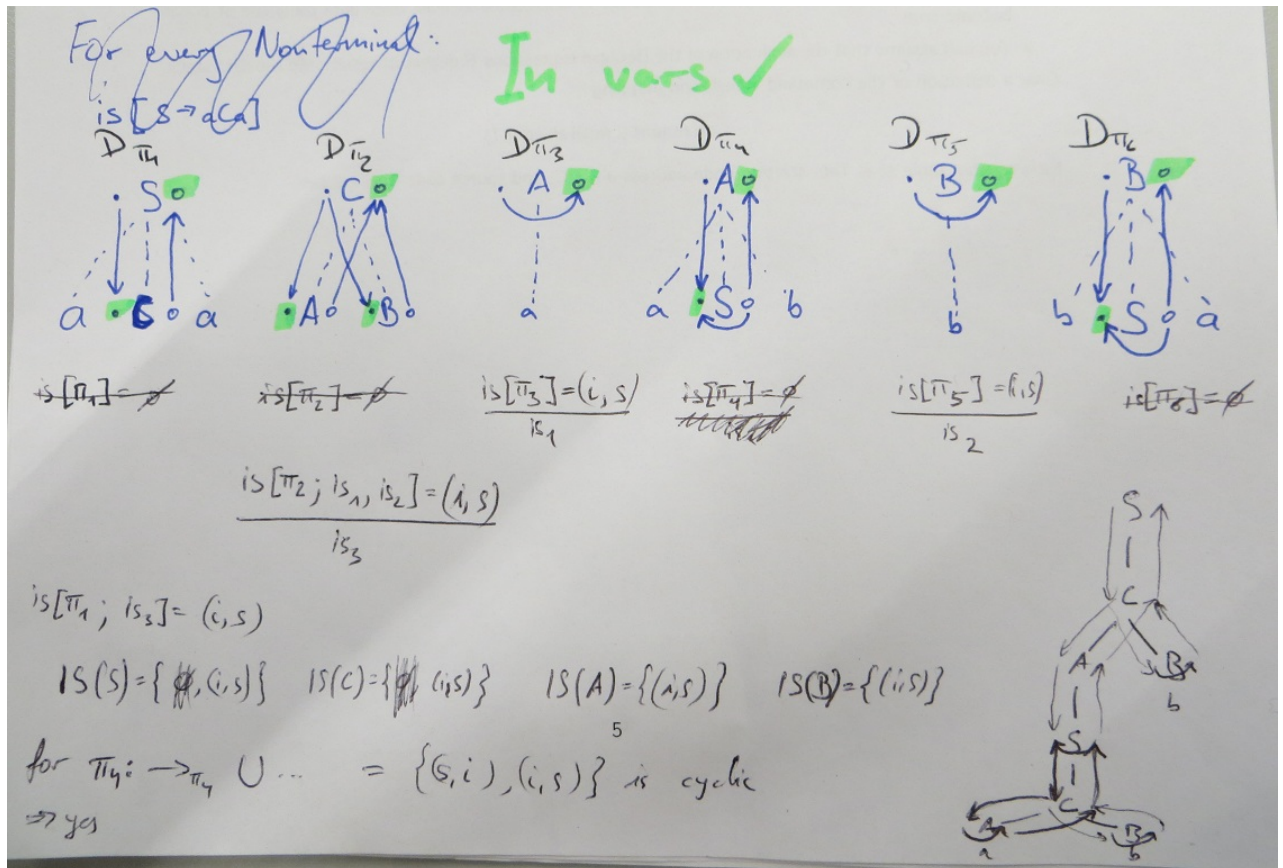
$$\begin{array}{lll}
S' & \rightarrow & S \quad s.0 = s.1 \\
S & \rightarrow & aCa \quad i.2 = i.0 \\
& & s.0 = s.2 \\
C & \rightarrow & AB \quad i.1 = i.0 \\
& & i.2 = i.0 \\
& & s.0 = s.1 + s.2 \\
A & \rightarrow & a \quad s.0 = i.0 + 1 \\
A & \rightarrow & aSb \quad i.2 = i.0 + s.2 \\
& & s.0 = s.2 \\
B & \rightarrow & a \quad s.0 = i.0 + 2 \\
B & \rightarrow & bSa \quad i.2 = i.0 + s.2 \\
& & s.0 = s.2
\end{array}$$

Apply the circularity test from the lecture in order to show that the attribute grammar is circular. To this aim,

- i) depict the dependency graph for each single production rule,
 - ii) iteratively construct $IS(X)$ for every nonterminal X , and
 - iii) construct a (partial) syntax tree with a cyclic dependency graph.
- b)** Keep the context-free grammar from above and provide a new (non-circular) attribution which computes the position of the first occurrence of b . E.g. for input word "aaaaabbaabaa", your grammar should yield the value 6 at the root node. If b does not occur in the given word w , the grammar should indicate this by a boolean flag or return a number ≤ 0 or $> |w|$.

Hint: You do **not** have to prove non-circularity.

Lösung:



a)

$S \rightarrow aCa$ $found.0 = found.1$
 $i.2 = 1$
 $s.0 = (found.2)?s.2 : s.2 + 1$

$C \rightarrow AB$ $found.0 = found.1 \vee found.2$
 $i.1 = i.0$
 $i.2 = s.1$
 $s.0 = (found.1)?s.1 : s.2$

b) $A \rightarrow a$ $found.0 = false$
 $s.0 = i.0 + 1$

$A \rightarrow aSb$ $found.0 = true$
 $s.0 = (found.2)?s.2 + i.0 + 1 : s.2 + i.0 + 2$

$B \rightarrow a$ $found.0 = false$
 $s.0 = i.0 + 1$

$B \rightarrow bSa$ $found.0 = true$
 $s.0 = i.0 + 1$

Question 5 (Code Generation):

(3 + 5 Points)

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

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

We assume that the corresponding code contains the following instructions:

```

:
3: LOAD (1, 1) % (dif, off)
4: LIT (1)
5: ADD
6: STORE(1, 1)
7: JMP(12)
8: LOAD(0, 2)
9: LOAD(0, 1)
10: SUB
11: STORE(0, 1)
12: RET
:

```

Determine the intermediate machine states during the execution of the program fragment above.

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

- b) Extend the code generation for EPL programs by taking post-testing loops into account. (Do this directly without an intermediate translation of repeat to while loops.)
- The syntax of EPL is extended by statements of the form

repeat C until B $\in Cmd$

where $B \in BExpr$ and $C \in Cmd$.

- The interpretation of this construct is as usual: C is executed at least once until the value of B has become true.
- You can assume that the evaluation of the Boolean expression B pushes its result onto the data stack.

Give a definition of the command translation mapping

ct(repeat C until B , st , a , l)

for a symbol table $st \in Tab$, starting code address $a \in PC$, and source code level $l \in Lev$.

Question 6 (Algorithmic Differentiation):**(7.5 + 7.5 Points)**

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

$$\begin{aligned} S &\rightarrow A \mid A S \\ A &\rightarrow v = E ; \\ E &\rightarrow E * E \mid v \end{aligned}$$

Nonterminal symbols denote assignments (A) and sequences thereof (S) in addition to arithmetic expressions (E). The start symbol is S . Terminal symbols represent scalar variables (v , for example, words over all lowercase letters), the assignment operator ($=$), the end of assignment marker ($;$), and binary floating-point multiplication ($*$).

- a) Give the LR(0) automaton of this grammar assuming leftmost resolution of associativity of binary floating-point multiplication.
- b) Extend the context-free grammar to an attribute grammar that generates tangent-linear derivative code. Use the binary floating-point operations “+” and “*” (addition and multiplication). Give the result of applying the corresponding single-pass compiler to the following program:

```
y=x*x*x;  
x=x*y;
```