

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

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 .

☐ True ☐ 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.

☐ True ☐ 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 .

☐ True ☐ 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.

☐ True ☐ Wrong

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

☐ True ☐ 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. $\mathcal{L}(G) = \mathcal{L}(G') = \mathcal{L}(G'') = \mathcal{L}(G''')$.

☐ True ☐ 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.

☐ True ☐ 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

☐ True ☐ 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

Lösung: _____

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

- ☐ True ☒ Wrong Corollary 3.9 and Example 3.7
- ☐ True ☒ Wrong Only in normal mode this is true, not in backtrack mode!
- ☒ True ☐ Wrong Since we have $LL(k) \subsetneq LL(k+1) \subsetneq \dots \subseteq LL$
- ☐ True ☒ Wrong There are k matching steps plus reduction steps!
- ☐ True ☒ Wrong lol
- ☒ True ☐ Wrong Slide 11.32
- ☐ True ☒ Wrong Checking just one tree is not enough. It does not matter whether all rules were used in general. All possible derivation trees must be circle free!
- ☒ IC ☐ MC Jasmin is a machine independent intermediate language.
- ☒ True ☐ Wrong True.

☐ 0 ☒ 1 This is the logic of the following *WHILE* program:

```

if (!(0==0 && 0==1)){
    write("1");
}
else{
    write("0");
}$

```

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

In this question we consider an XML-like language that shall describe 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 $\Sigma = \{a, \dots, z, 0, \dots, 9, \sqcup, <, >, =, +, -, *\}$ for the following symbol classes:
- opening tag (A tag with some name consisting of letter 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) Argue why or why not!
- 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?
- c) Assume we implement the FLM principle using a Backtracking DFA. Give an example scenario in which it has to backtrack and explain how the input word affects the backtracking depth¹!
- d) Can you modify any of the above regular expressions such that
- the performance of the Backtracking DFA is improved, and
 - the language (though slightly modified) remains as expressive as the original²?

Lösung: _____

- a)
- tag = $\langle id \rangle$
 - ctag = $\langle /id \rangle$
 - string = $(a + \dots + z + \sqcup)^+$
 - constraint = $('-' + \epsilon) \alpha (('-' + '+') \alpha)^* op (num + id)$

where

- $id = (a + \dots + z)^+$
- $num = (0, \dots, 9)^+$
- $\alpha = num + id + num * id$

¹Backtracking depth is the number of steps the automaton runs in the “wrong” direction until it realises it needs to go back.

²For every word in the original language there is a word in the modified language such that, mathematically speaking, they describe the same system of linear inequalities.

- $op = < + < '=' + > + > '=' + '='$

b) • It does not. This because the languages of the above regular expressions are pair-wise distinct. A longest match is therefore unique and no precedence between the expressions is required for a successful analysis.

- No. Consider the string

$$x > 0$$

If string precedes constraints, you cannot match the '>'. If constraint precedes string, string will match 'x' and the remaining '> 0' cannot be matched. Thus, no analysis is possible.

c) Note that we cannot assume well-formed XML structure of the input as only the parser can ensure this (in a later stage).

On input like `<a>b<a>`, the Backtracking DFA would try to match a constraint $b < a$ before it sees the ">" which cannot be part of the current constraint nor the beginning of a new symbol and then it backtracks.

The longer the name of a tag is, the longer it needs to find out, it has to backtrack. E.g. for `<a>b<aaaaaaaaa>` we have a depth of 10.

d) Yes, forbid variable names on the right hand side of constraints, then the backtracking depth is limited to 1 no matter what the input is.

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

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

- 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, $A_1A_2 \cdots A_n$ 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 $k \in \mathbb{N}$.

Lösung:a) • From a theorem in the lecture we know that $G \in LL(1)$ iff

$$\text{la}(A \rightarrow \beta) \cap \text{la}(A \rightarrow \gamma) = \emptyset \quad \text{for } \beta \neq \gamma$$

However,

$$\text{la}(\text{Cmd} \rightarrow N) \cap \text{la}(\text{Cmd} \rightarrow P) = \{ \} \neq \emptyset .$$

Thus the grammar is not $LL(1)$!

- Apply left factoring, pushing the “[” all the way up to S productions. So G' becomes:

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

CHECK THIS COINCIDES WITH THE METHOD FROM THE LECTURE!!!

b) This statement is wrong! Consider the following grammar:

$$\begin{array}{lcl} A & \rightarrow & a B \mid a C \\ B & \rightarrow & a B \mid b \\ C & \rightarrow & a C \mid c \end{array}$$

For every k , take input $a^k b$ or $a^k c$. Then a deterministic top-down parser with lookahead k cannot decide which rule to start with. Formally, we consider the $LL(k)$ definition for any k :

$$A \Rightarrow_l^0 A \begin{cases} \Rightarrow a B \Rightarrow_l^* a^k b \\ \Rightarrow a C \Rightarrow_l^* a^k c \end{cases}$$

We have $w = \varepsilon$, $\alpha = \varepsilon$, $x = a^k b$, $y = a^k c$, $\beta = aB$ and $\gamma = aC$. Although $\beta \neq \gamma$, we have $first_k(x) = first_k(y)$. Thus the grammar is not in $LL(k)$ for any k .

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

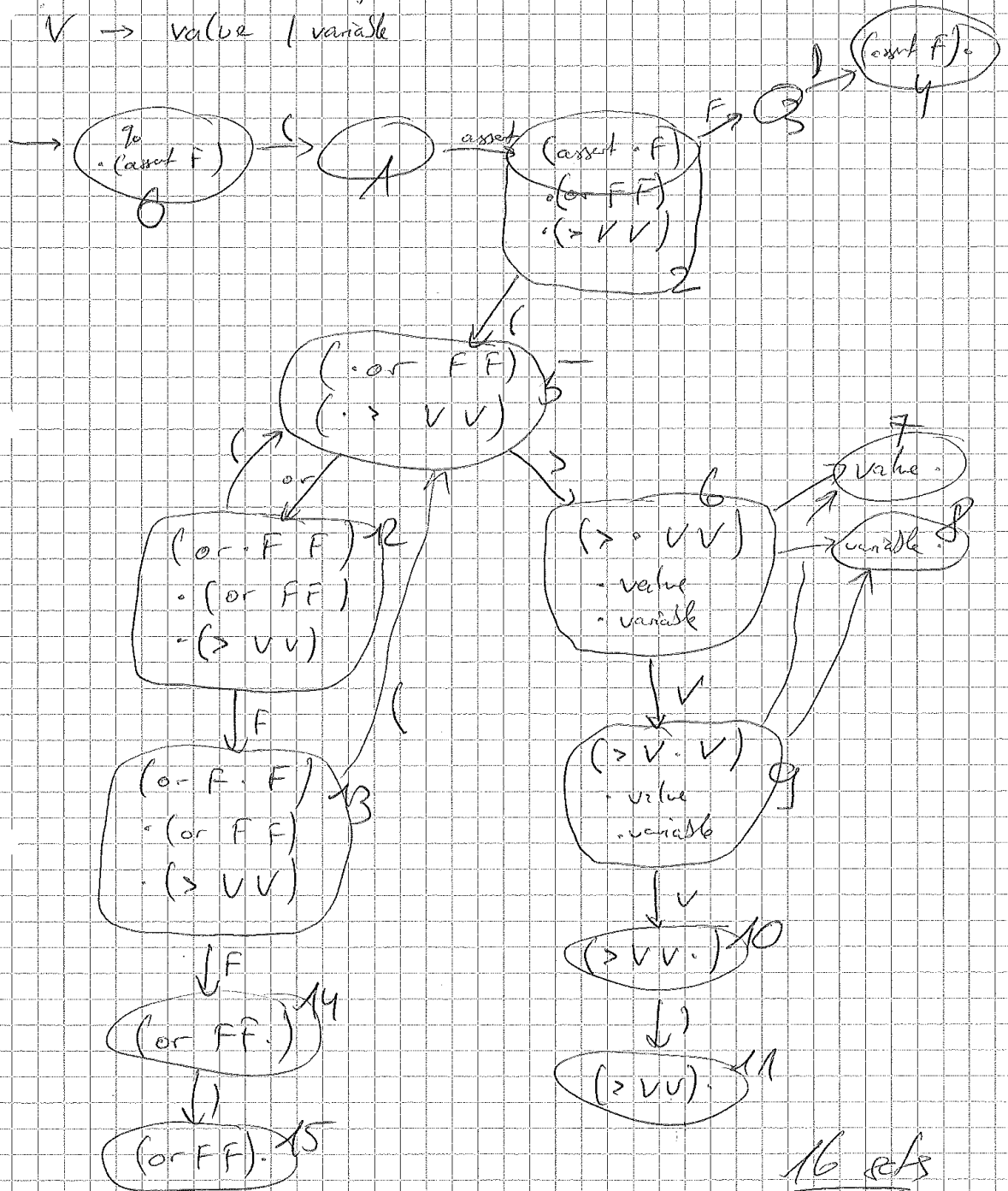
Consider the following grammar G :

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

- 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 $w = (\text{assert } (> 5 4))$.

Lösung: _____

$A \rightarrow (\text{assert } F)$
 $F \rightarrow (\text{or } F F) \mid (> V V)$
 $V \rightarrow \text{value} \mid \text{variable}$



Compute LR(0) sets, transitions (or depict Goto DFA)
 Why is this grammar a LR(0) grammar?

a)

- b)** In each $LR(0)$ set there are no conflicts. You can always either shift or reduce with exactly one rule.
- c)** We use the notation $\langle word, stack, analysis \rangle$. Starting with $w = (\text{assert}(> 5\ 4))$ we get the following run:

$$\begin{aligned}
 &\langle w, l_0, \epsilon \rangle \vdash \langle \text{assert}(> 5\ 4)), l_0 l_1, \epsilon \rangle \\
 &\quad \vdash \langle (> 5\ 4)), l_0 l_1 l_2, \epsilon \rangle \\
 &\quad \vdash \langle > 5\ 4)), l_0 l_1 l_2 l_5, \epsilon \rangle \\
 &\quad \vdash \langle 5\ 4)), l_0 l_1 l_2 l_5 l_6, \epsilon \rangle \\
 &\quad \vdash \langle 4)), l_0 l_1 l_2 l_5 l_6 l_7, \epsilon \rangle \\
 &\quad \vdash \langle 4)), l_0 l_1 l_2 l_5 l_6 l_9, V \rightarrow \text{value} \rangle \\
 &\quad \vdash \langle \rangle), l_0 l_1 l_2 l_5 l_6 l_9 l_7, V \rightarrow \text{value} \rangle \\
 &\quad \vdash \langle \rangle), l_0 l_1 l_2 l_5 l_6 l_9 l_{10}, V \rightarrow \text{value}; V \rightarrow \text{value} \rangle \\
 &\quad \vdash \langle \rangle, l_0 l_1 l_2 l_5 l_6 l_9 l_{10} l_{11}, V \rightarrow \text{value}; V \rightarrow \text{value} \rangle \\
 &\quad \vdash \langle \rangle, l_0 l_1 l_2 l_3, F \rightarrow (> V V); V \rightarrow \text{value}; V \rightarrow \text{value} \rangle \\
 &\quad \vdash \langle \epsilon, l_0 l_1 l_2 l_3 l_4, F \rightarrow (> V V); V \rightarrow \text{value}; V \rightarrow \text{value} \rangle \\
 &\quad \vdash \langle \epsilon, l_0, A \rightarrow (\text{assert } F); F \rightarrow (> V V); V \rightarrow \text{value}; V \rightarrow \text{value} \rangle
 \end{aligned}$$

The automaton accepts and returns the rightmost derivation of the word.

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

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

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

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?

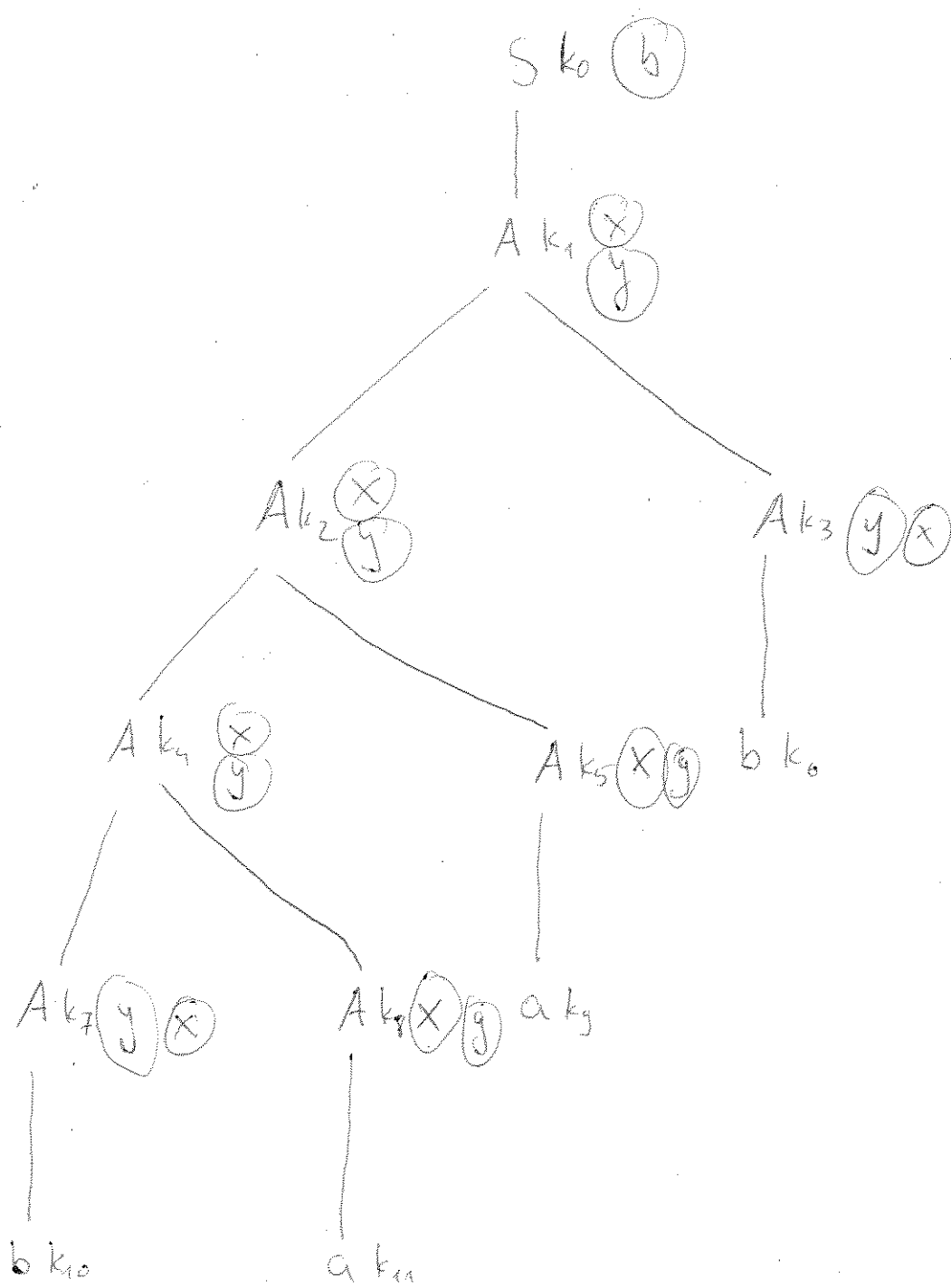
b) Now consider another attribute grammar:

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

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.

Lösung: _____



a)

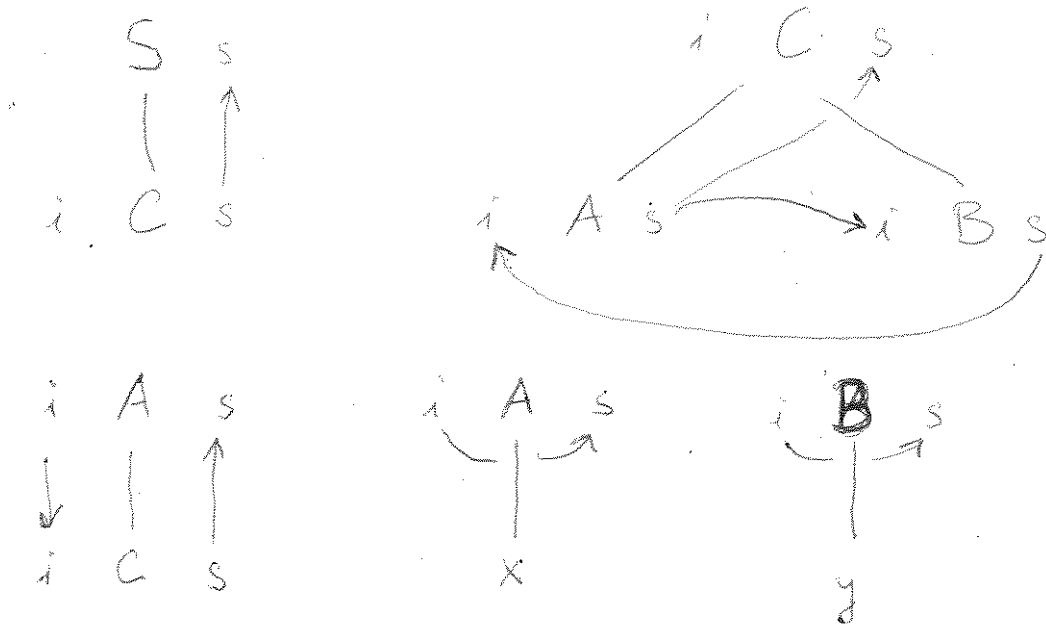
The corresponding system of equations is

$$\begin{aligned}
 y.k_7 &= 1 \\
 x.k_8 &= 1 \\
 x.k_5 &= 1 \\
 y.k_3 &= 1 \\
 x.k_7 &= 0 \\
 y.k_8 &= 0 \\
 y.k_5 &= 0 \\
 x.k_3 &= 0 \\
 x.k_4 &= x.k_7 + x.k_8 \\
 y.k_4 &= y.k_7 + y.k_8 \\
 x.k_2 &= x.k_4 + x.k_5 \\
 y.k_2 &= y.k_4 + y.k_5 \\
 x.k_1 &= x.k_2 + x.k_3 \\
 y.k_1 &= y.k_2 + y.k_3 \\
 b.k_0 &= x.k_1 == y.k_1
 \end{aligned}$$

The unique solution is

$$\begin{aligned}
 y.k_7 &= 1 \\
 x.k_8 &= 1 \\
 x.k_5 &= 1 \\
 y.k_3 &= 1 \\
 x.k_7 &= 0 \\
 y.k_8 &= 0 \\
 y.k_5 &= 0 \\
 x.k_3 &= 0 \\
 x.k_4 &= 1 \\
 y.k_4 &= 1 \\
 x.k_2 &= 2 \\
 y.k_2 &= 1 \\
 x.k_1 &= 2 \\
 y.k_1 &= 2 \\
 b.k_0 &= \text{true}
 \end{aligned}$$

It is checked whether the given word contains equally many *a*'s and *b*'s.



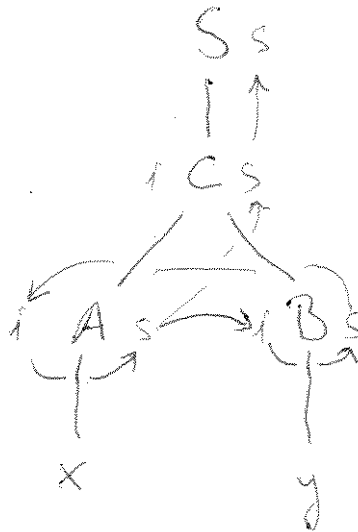
b)

- $is_1 = is[B \rightarrow y] = \{(i, s)\}$
- $is_2 = is[A \rightarrow x] = \{(i, s)\}$
- $is_3 = is[C \rightarrow AB, is_1, is_2] = \emptyset$
- $is_4 = is[A \rightarrow C, is_3] = \emptyset$
- $is_5 = is[C \rightarrow AB, is_4, is_2] = \emptyset$
- $is_6 = is[S \rightarrow C, \emptyset] = \emptyset$

This gives us:

- $IS(S) = \emptyset$
- $IS(C) = \emptyset$

- $IS(A) = \{(i, s)\}$
- $IS(B) = \{(i, s)\}$



The grammar is circular!

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;
.
.
.

```

Lösung:

- a)
- ```

1: CALL(main,0,0);
2: JMP(0);
P: LOAD(2,1);
4: LIT(2);
5: MUL;
6: STORE(2,1);
7: RET;
Q: LOAD(2,1);
9: LIT(1);
10: ADD;
11: STORE(2,1);
12: RET;
main: LOAD(1,1);
14: LIT(1);
15: GEQ;
16: JFALSE(else);
17: CALL(P,0,0);

```

```
18: JMP(end);
else: CALL(Q,0,0);
end:
21: RET;
```

**b)**

|                 |                                                     |
|-----------------|-----------------------------------------------------|
|                 | (20, $d : 6, 3 : 2 : 28 : 4 : 3 : 2 : 17 : p$ )     |
| 20: LIT(7);     |                                                     |
|                 | (21, $d : 6 : 7, 3 : 2 : 28 : 4 : 3 : 2 : 17 : p$ ) |
| 21: MUL;        |                                                     |
|                 | (22, $d : 42, 3 : 2 : 28 : 4 : 3 : 2 : 17 : p$ )    |
| 22: STORE(1,1); |                                                     |
|                 | (23, $d, 3 : 2 : 28 : 4 : 3 : 2 : 42 : p$ )         |
| 23: RET;        |                                                     |
|                 | (28, $d, 4 : 3 : 2 : 42 : p$ )                      |

## 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) \\
 \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.