Examination

Programming Languages and their Compilers

Hints:

1) Fill in your name and signature before you start:

Name: First Name: Signature:

2) The tasks are to be solved on the according leaf. Only if there isn't enough space (on front and back side) you are allowed to use the empty pages at the end. In those cases give a hint on the according task page. You aren't allowed to use your own paper.

3) Unbound pages are forbidden strictly. Don't remove the tacking of the examination. **No tools** are allowed. The **operation time** is **105 minutes**.

4) **Overview:**

Task	Content	Points
1	Regular Expressions/Definitions, Transition Diagrams	11
2	Context Free Grammar, Ambiguity, Top Down Problems	7
3	Syntax, Transition Diagrams, Code	11
4	Bottom Up Parsing	4
5	Syntax Directed Type Checking/Counting of Expressions	8
6	Top Down Translation Scheme	7
7	Bottom Up Translation Scheme	6
8	Runtime Stack, Static Chains, Display	8
9	Syntax Directed Generation of Intermediate Code	6
10	Global Data Flow Analysis, Optimization	11
11	Code Generation using a DAG	7
	Total	86

Addition of points:

Mark:

Task 1: (5 + 2 + 4 Points)

a) Assume the following regular expression to be given

(a* b | a c)* a b* (a* | b* c)

Give a representation of this expression by **one single** deterministic transition diagram.

b) A programming language has the following kinds of lexical elements:

- **Constants**: non-empty sequence of lower case letters and digits starting with a letter
- **Variables**: non-empty sequence of letters and digits starting with a letter and containing at least one capital letter
- **Numbers**: non-empty sequence of digits, thereby 'leading 0's are not allowed
- **Other**: '(' , ')' , ','

How do regular definitions for these lexical elements look like?

c) Represent the regular definitions of b) with **one single** deterministic transition diagram. Thereby the categories **const, var, num, opa, cpa** and **com** have to be recognised.

Task 2: (5 + 2 Points)

Assume the following rules of a context free grammar to be given

```
S    ->    a  S  b
S    ->    a  S
S    ->    b  a
```

a) Is this grammar ambiguous? Prove your answer!

b) Which problems arise if you try to implement this grammar with a top down method? Solve these problems!

Task 3: (3 + 4 + 4 Points)

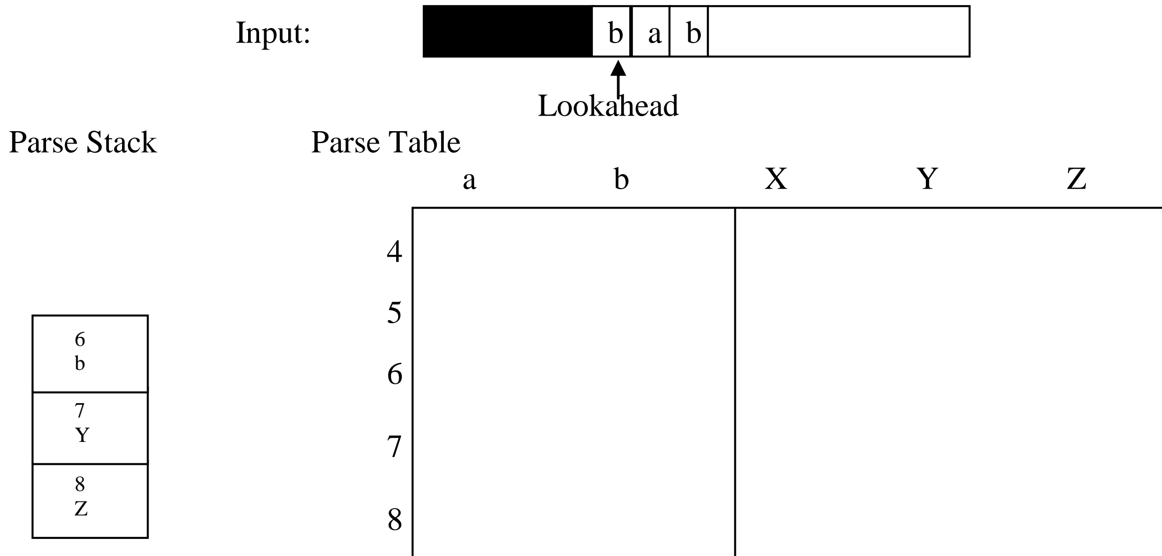
a) Define rules of a context free grammar (top-down!) to define a predicate. It starts with a predicate name (**const**), optionally followed by an opening parenthesis (**opa**), a non-empty list of terms (separated by **com**) and a closing parenthesis (**cpa**). A term can be a **const**-value, a variable(**var**) or a function call, that looks like a logical predicate. Assume the highlighted tokens to be returned by a Scanner.

b) Represent the syntax rules defined in a) by transition diagrams. Simplify and combine these diagrams as far as possible.

c) Encode the diagrams of b) (in a PASCAL-like notation) using the method ‘recursive descent’. The procedure SCANNER(X) can be used to get the next lexical element into the global variable X.

Task 4: (4 Points)

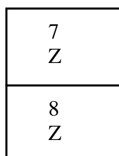
A snapshot during a LR-parsing process is given by the content of the parse stack and the current position of the Lookahead. The following 3 steps are pointed out by the resulting modifications on the parse stack.



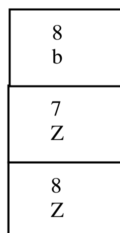
Insert the needed information into the parse table that caused these modifications. Additionally define the needed grammar rules.

Grammar Rules:

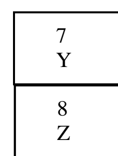
Stack after 1st action



Stack after 2nd action



Stack nach 3rd action



Task 5: (8 Points)

Add to the grammar rules defined in task 3 semantic actions for checking the number of parameters of a predicate and for calculating the total number of **const**-values contained in the term list of a predicate. Thereby only the types **const** and **var** are allowed. The number of parameters of a predicate as well as of a function call can be looked up by getpars(…..).

Task 6: (7 Points)

Assume the following sequence of a translation scheme

$S \rightarrow a \; \{S_1.B := S.B\} \; S_1 \; b \; \{S.A := S_1.A\}$
$S \rightarrow b \; \{S_1.B := S.B\} \; S_1 \; b \; \{S.A := S_1.A + 2\}$
$S \rightarrow a \; c \; \{S.A := S.B\}$

Give an implementation of this sequence (in PASCAL-like notation) using the method 'recursive descent'. A call of SCANNER(X) can be used to get the next lexical element into the global variable X.

Task 7: (6 Points)

Assume the following sequence of a translation scheme

S -> A B C D E	{ C.i := A.s + B.s; D.i := C.s; E.i := D.i + C.i }
S -> B C D E	{ C.i := B.s; D.i := C.s + B.s; E.i := D.s }
C -> c	{ C.s := C.i }
D -> ε	{ D.s := D.i + 1}
E -> a	{ E.s := E.i + 1 }

Describe the modifications that are necessary if you try to implement it as a bottom up translation scheme.

Task 8: (4 + 4 Points)

A program is given having the following structure:

program P;
    function A(function X …….): ....;
    begin ......X(D)....... { **now** *} end;*
    function B(function X ....): ......;
        function C (function X ....): ......;
        begin ....X(A).... { **here** *} end;*
    begin .... X(C) .... end;
    function D(function X ....): ......;
        function E (function X ....): ....;
        begin ....X(B)......... { **then** *} end;*
    begin ....X(E).......... end;
begin ....B(D)........end.

a) How does the runtime stack in the situation **now**, **here** and **then** look like? Don't forget to figure out the static chains.

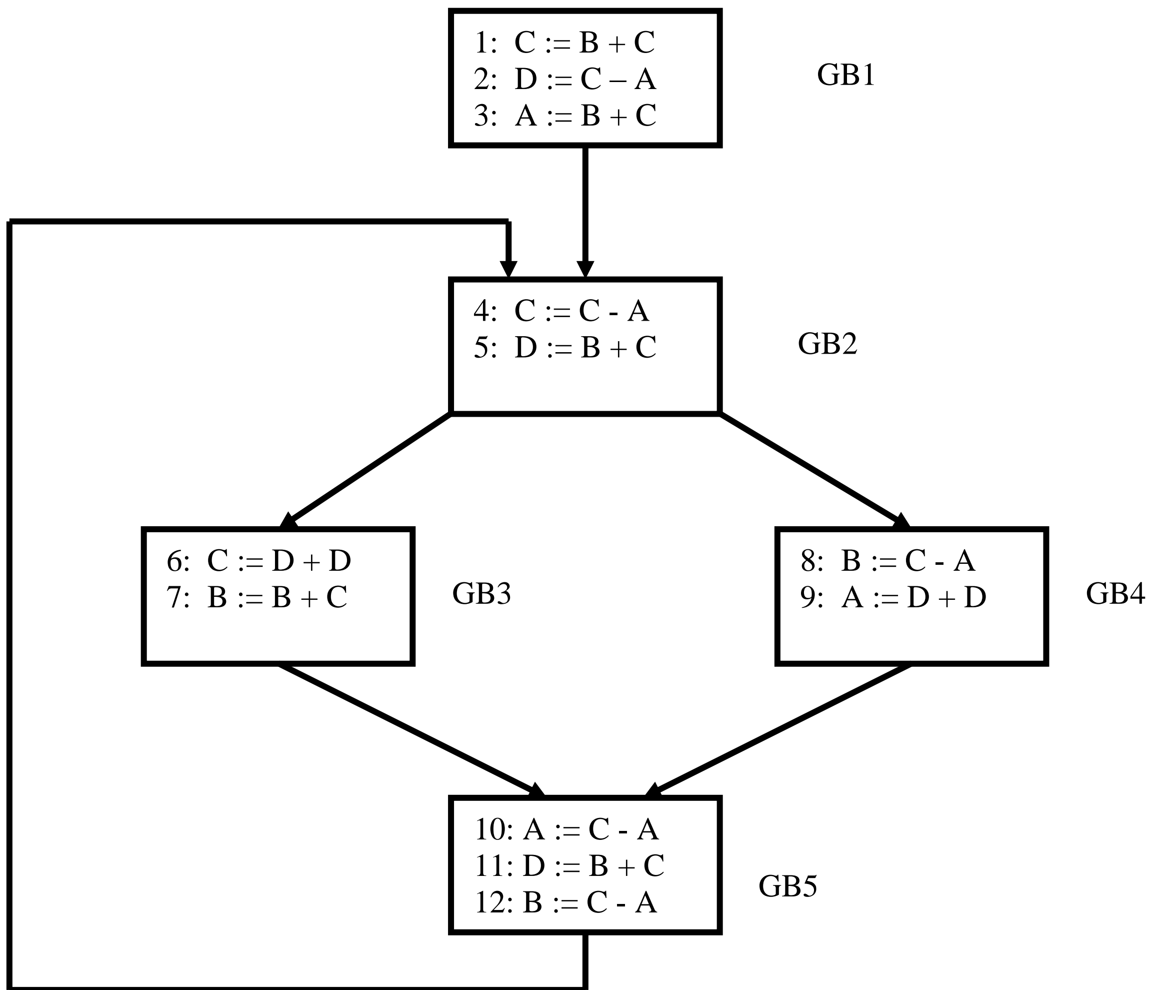
b) Describe the three situations mentioned above if a display technique is used.

Task 9: (6 Points)

Extend the syntax directed bottom up translation scheme defined during the semester by grammar rule(s) for a **record**-declaration (as defined in programming language PASCAL) together with semantic actions for determining the types and relative addresses of the components and insert them into a (local) symbol table. Use thereby the stacks TAB_PTR and REL_ADR as well as the operations *createtab* and *addwidth* as defined in the lecture.

Task 10: (2 + 4 + 2 + 3 Points)

Assume the following flow graph to be given



a) Which kind of optimization is possible in this section? Which data flow problem has to be solved therefore?

b) According to the problem mentioned in a) define for each basic block GB the sets GEN(GB) and DEL(GB).

c) Which initialisation is generated in this case by the iterative algorithm for the basic blocks of the flow graph?

d) State which improvements are possible within this sequence and which information according to the sets resulting from the iterative algorithm lead to these improvements.

Task 11: (3 + 1 + 3 Points)

Assume a basic block contains the following code

```
a := a - b;
c := c + b;
b := a - b;
a := b + c;
b := a - b;
c := a - b;
```

a) Give the according DAG-representation of this code.

b) Which kinds of optimization are possible in this block?

c) Generate (efficient) machine code out of this DAG. Thereby the registers R1 and R2 are available. Use the following statement types:

MOVE a, b Load (or store) content of a to b
ADD a, b Add content of a to the content of b, the result will be contained in b (i.e. b:=b+a)
SUB a, b Subtract content of a from the content of b, the result will be contained in b (b:=b-a)
ADD and SUB actions have to be executed within a register!