Final Exam
Compiler Design
Dozenten: Dr. Biagio Cosenza, Dr. Nicolai Bull, Prof. Ben Juurlink

22.02.2019

Fakultät IV
Institut für Technische Informatik und Mikroelektronik
FG Architektur eingebetteter Systeme

First Name : .....

Last Name : .....

Matrikelnummer : .....

Course of study : .....

: ◯ EIT ◯ Erasmus

Exercise	1	2	3	4	5	$\sum$
max. points	18	4	24	6	6	58
reached points						
corrector						

Important Instructions:

- Fill out the top of the sheet with your name, matriculation number and other data.
- The exam takes 89 minutes.
- For each exercise, you can see how many points can be achieved, for a total of 50 points.
- Write in comprehensible English, only what is asked, and highlight the final result.
- You are not allowed to use laptops, smart-phones, smart-watches and any other similar devices.
- You can not have anything but your ID, a pen, and this exam on your desk.
- Switch off your phones and other noisy gadgets.
- You can not talk to anyone during the exam.
- Cheating is punishable by a failure to pass the whole module (including the lab).

1. Exercise: Lexical Analysis (18 points)

1.1 Regular Expression (8 points)

Write a Regular Expression for the following languages:

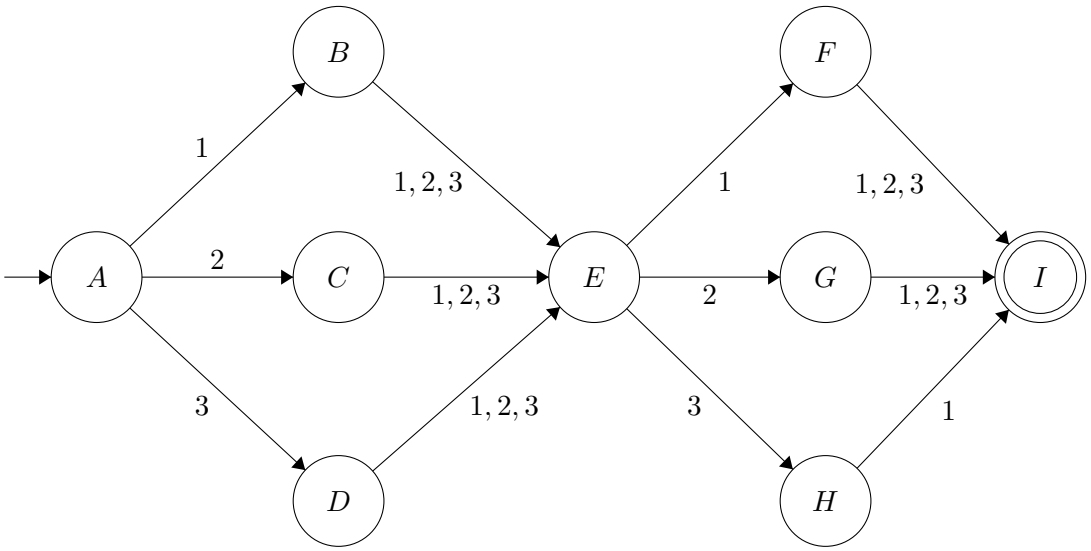
- $L = \{a^n b^m | n \geq 4, m \leq 3\}$
- $L = \{a^n b^m | (n + m) \text{ is even}\}$
- $L = \{w \in \{0, 1\}^* | w \text{ has at least one pair of consecutive zeros}\}$
- $L = \{w \in \{0, 1\}^* | w \text{ has no pair of consecutive zeros}\}$

1.2 DFA Minimization (10 points)

Use Hopcroft's Algorithm to minimize the states of the following DFA. You should

(a) fill the table with all iterations of the algorithm

(b) write the final DFA



Iteration	Groups	Split on 1	Split on 2	Split on 3

2. Exercise: Syntax and Semantic Analysis (4 points)

2.1 Expression Grammar (4 points)

(a) Write a grammar matching simple arithmetic expressions

- supporting the operators `+`, `-`, `*` and `/`,
- supporting parenthesized expressions (like `(1+2)*3`),
- using `num` as the number terminal,
- and taking into account precedence (`*` and `/` have higher precedence than `+` and `-`) and associativity (left to right), in the sense that the resulting parse tree should exhibit correct grouping.

(b) Is this language regular? Justify your answer. (A formal proof is not necessary.)

3. Exercise: Intermediate Representation (24 points)

3.1 Very Busy Expressions (10 points)

An expression *a op b* is said to be *very busy* at program point P if along every control flow path from P there is an expression *a op b* before a redefinition of a or b.

Show how very busy expressions may be calculated using iterative dataflow analysis by filling out the following table. The *Dominators* analysis is provided as a reference for the expected format. **If you use gen and kill sets, don't forget to precisely define what they contain for this analysis**.

	Dominators	Very Busy Expression
Domain	Sets of blocks	
Direction	Forward	
Meet	Intersection	
Transfer Func.	$f_B(x) = x \cup \{B\}$	
Equations	$\text{OUT}[B] = f_B(\text{IN}[B])$ $\text{IN}[B] = \bigcap_{P \in pred(B)} \text{OUT}[P]$	
Initialization	$\text{OUT}[P] =$ all blocks	
Boundary	$\text{OUT}[Entry] = \emptyset$	

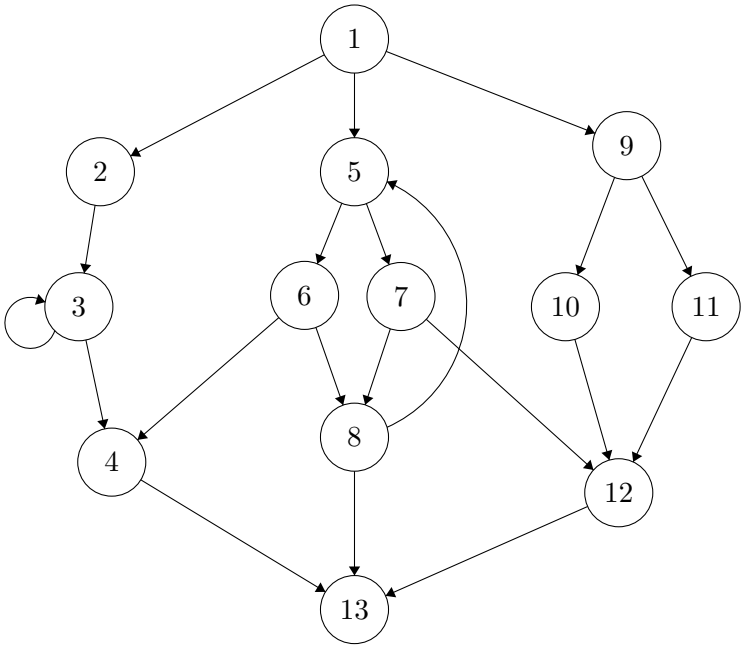
3.2 Control Flow Graph and Dominator Tree (14 points)

Given the following Control-Flow-Graph

(a) What nodes are dominated by node 5?

(b) What nodes are strictly dominated by node 5?

(c) Draw the dominator tree.



4. Exercise: Runtime, Code Generation, Registry Allocation (6 points)

4.1 Registry Allocation (6 points)

Given the code:

```
1  b = a + 2;
2  c = b * b;
3  b = c + 1;
4  return b * a;
```

(a) Write the set of live variables before each instruction (i.e., $IN(i)$).

(b) Draw the register interference graph.

(c) Assume to have only two physical registers, find an optimal registry allocation.

5. Exercise: Instruction Scheduling, Code Transformations (6 points)

5.1 Optimization (6 points)

The original code shown on the left has been transformed into the code on the right by inlining the `muldiv` function into the `test` function. Starting from the inlined variant, name and apply at least three additional optimizations. Show the new code after each applied optimization.

```
/// Original code
static unsigned muldiv(
    unsigned arg1, unsigned arg2,
    unsigned op) {
  if (op == 0) {
    return arg1 * arg2;
  } else {
    return arg1 / arg2;
  }
}
unsigned test(unsigned arg) {
  return muldiv(arg, 16, 1);
}
```

```
/// After inlining
unsigned test(unsigned arg) {
  unsigned arg = arg1, arg2 = 16, op = 1;
  if (op == 0) {
    retval = arg1 * arg2;
  } else {
    retval = arg1 / arg2;
  }
  return retval;
}
```