Examination 2021-01-14

Kompilatorteknik 1 (1DL321) – 5 hours

This is an examination **done at home**, so you can have your books and notes open when you take it. Please use an editor or word processor for all your answers in text. For questions that require drawing some picture or graph, you can either use an appropriate program or draw them on paper and attach a (good quality) picture of your solution to your answers.

When you finish, you need to either **upload a (preferably single) PDF** to your group in Studium or, in case you have trouble with uploading it, submit your PDF to the teacher. In either case, **you are supposed to stop answering questions at 13:00**, and your submission **must be uploaded/received by 13:15**. Remember to **write your anonymous code** in all PDF documents you submit.

The exam contains **40** points in total and their distribution between sub-questions is clearly identifiable. To get a final grade of **5** you must score at least **34**. To get a grade of **4** you must score at least **29** and to get a final grade of **3** you must score at least **20**.

Whenever in *real doubt* for what a particular question might mean, make sure to **state your assumptions clearly**. During the duration of the exam, you can send questions to the teacher either **by sending him a mail**, or **by calling him on the mobile phone** that you can find in his homepage.

DON'T PANIC!

1. **Automata and Regular Expressions (3 + 1 = 4 pts total)** Consider a language with alphabet $\Sigma = \{\blacktriangledown, \blacktriangle, \bullet\}$. If α is a string over Σ, we define as the *measure* of α (and write $|\alpha|$) the integer that gets computed as follows:

   - the measure of the empty string is equal to 0;
   - the $\bullet$ symbols do not affect the measure of a string;
   - each $\blacktriangle$ symbol adds one to the measure of a string; and
   - each $\blacktriangledown$ symbol subtracts one from the measure of a string.

   For example, $|\bullet\blacktriangle\bullet\blacktriangledown\blacktriangle| = 1$ and $|\blacktriangledown\bullet\blacktriangledown| = -2$. The language L consists of all strings over Σ that satisfy the following constraint: each of its substrings has a measure whose absolute value is less than or equal to 2. For example, the strings $\blacktriangle\blacktriangle, \blacktriangle\bullet\bullet\blacktriangledown\blacktriangle, \bullet\blacktriangledown\bullet\blacktriangle\blacktriangle\blacktriangledown, \blacktriangle\bullet\blacktriangle\blacktriangledown\blacktriangledown\bullet\bullet\blacktriangle\blacktriangledown$ and $\blacktriangle\blacktriangledown\blacktriangle\bullet\blacktriangledown\blacktriangledown\blacktriangle\bullet\blacktriangledown\blacktriangle\blacktriangle$ belong to L, while the strings $\blacktriangle\blacktriangle\bullet\blacktriangledown\blacktriangle\blacktriangle$ and $\blacktriangledown\blacktriangle\blacktriangle\bullet\blacktriangledown\blacktriangle\bullet\blacktriangle\blacktriangledown\blacktriangledown$ do not (the first has measure 3, while in the second the substring $\blacktriangle\blacktriangle\bullet\blacktriangledown\blacktriangle\bullet\blacktriangle$ has measure 3).

   (a) **(3 pts)** Design a finite automaton, in the form of a transition graph, that recognizes the strings of language L.

   (b) **(1 pt)** Is the automaton of your answer to the question above deterministic or not? Briefly explain.

2. **LL Grammars (7 pts total).** Consider the following context-free grammar for a language of expressions where `id` denotes an identifier (e.g. `a, b`, ..., `x, y, z`), and the assignment (`=`), addition(`+`), and multiplication (`*`) operators have the same meaning as in C/C++:

$$\begin{aligned} E &\rightarrow \texttt{id} = E \mid T\ T' \\ T' &\rightarrow \epsilon \mid \texttt{+}\ T\ T' \\ T &\rightarrow F\ F' \\ F' &\rightarrow \epsilon \mid \texttt{*}\ F\ F' \\ F &\rightarrow \texttt{(}\ E\ \texttt{)} \mid \texttt{id} \end{aligned}$$

   (a) **(1 pt)** Explain why this grammar is not LL(1).

   (b) **(2 pts)** Construct an equivalent LL(1) grammar.

   (c) **(3 pts)** Construct the LL(1) parsing table for the grammar you gave as answer to the previous question.

   (d) **(1 pt)** Use the table of your answer to recognize the string `x = y + (z = a * b)`.

---

3. **LR Parsing (8 pts total).** Consider the following context-free grammar:

$$S \rightarrow \texttt{id} \mid \texttt{string} \mid S\ \texttt{+}\ S \mid \texttt{id}\ \texttt{:=}\ S$$

   (a) **(1 pt)** Show that this grammar is ambiguous.

   (b) **(5 pts)** Construct the DFA of LR(1) items. Recall that you first need to augment the grammar with a new rule $S' \rightarrow S\ \$$.

   (c) **(2 pts)** Which states of the DFA have conflicts and of what kind (shift-reduce or reduce-reduce)? Be sure to state clearly the cause of the conflict in each case.

---

4. **Code Generation and Optimization (4 pts total).** Consider the following piece of C code and assume that the only variables that are live at its end are `x` and `y`.

```
n = 0, m = 1;
c = m * b;
d = c * a;
x = d + (a + n) * b;
y = (7 + m - n) * d;
```

   (a) **(1 pt)** Give the three-address code that will be produced for this code without applying any optimizations.

   (b) **(3 pts)** Apply as many optimizations as possible to the code of your answer to the above question: algebraic simplification, constant propagation and constant folding, common subexpression elimination, copy propagation, and dead code elimination. At each step, identify the optimization that gets applied. You can apply the optimizations in any order. You need not show the complete code at each step, as long as it is clear which part of the code gets optimized every time.

---

5. **Syntax Errors (4 pts).** Give an example of a syntactically erroneous program fragment in which consideration of semantic information (e.g., types) might help a compiler make a good choice between two plausible "corrections" of the input. Briefly explain your answer.

---

6. **Scoping and Parameter Passing Mechanisms (2 + 4 = 6 pts total).** Consider the two programs below in some hypothetical programming language.

```
int x = AC4;
void g(int a, int b) {
  print(a, x);
  x = b;
}
void f(int y) {
  int x = AC3;
  g(x, y);
  print(x);
}
void main() {
  f(42);
  print(x);
}
```

```
void p(int a, int b, int c) {
  b = b + 5;
  b = a + c + 4;
  print(a, b, c);
}

void main() {
  int j = AC4;
  int k = AC3;
  p(j, j, j + k);
  print(j, k);
}
```

In these programs, AC_3 and AC_4 denote the last two digits of the numeric part of your anonymous code for this exam. For example, if your anonymous code is AN-0042-XYZ, then $AC_3 = 4$ and $AC_4 = 2$.

(If you do not have such an anonymous code, use $AC_3 = 4$ and $AC_4 = 2$ for your answers.)

Your task is to answer what these programs will print in a variety of situations.

(a) The program on the left if the language uses static scoping.

(b) The program on the left if the language uses dynamic scoping.

(c) The program on the right if all parameters are passed by value.

(d) The program on the right if `a` and `b` are passed by reference and `c` by value.

(e) The program on the right if `a` and `b` are passed by value-result and `c` by value.

(f) The program on the right if all parameters are passed by name (call by name).

Provide your answers in a table like the following one:

	$AC_3 =$		$AC_4 =$		
(a)	`a =`	`x =`	`x =`	`x =`	
(b)	`a =`	`x =`	`x =`	`x =`	
(c)	`a =`	`b =`	`c =`	`j =`	`k =`
(d)	`a =`	`b =`	`c =`	`j =`	`k =`
(e)	`a =`	`b =`	`c =`	`j =`	`k =`
(f)	`a =`	`b =`	`c =`	`j =`	`k =`

---

7. **Basic Blocks, Liveness Analysis, and Register Allocation (7 pts total).** Consider the following fragment of intermediate code:

```
L0: x := 1
    a := load 0($fp)
    b := load 4($fp)
L1: p := load 0(a)
    q := p + x
    r := load 0(b)
    q := q - r
    if p = r goto L3
L2: b := b + 4
    goto L4
L3: x := 0
    a := a + 4
    goto L1
L4: x := q * r
    if b < a goto L3
L5:
```

(a) **(1 pts)** Break the above piece of code into basic blocks and draw its control-flow graph. Place each basic block in a single node.

(b) **(3 pts)** Annotate your control-flow graph with the set of variables live before and after each statement (not just before and after every block!), assuming that only `x` is live at label `L5`. Make sure it is clear where your annotations are placed.

(c) **(3 pts)** Draw the register inference graph for this intermediate code and assign each temporary to a register for a machine that has four available registers (`$r0-$r3`). Note that you cannot use `$fp` for temporaries. If you need to spill some temporary into memory, rewrite the program using pseudo-instructions `load y` and `store y` to load and spill the value of temporary `y` from and to memory.

Good luck !