Linköpings universitet
IDA Department of Computer and Information Sciences
Prof. Dr. Christoph Kessler

# TENTAMEN / *EXAM*

## **TDDB29** Kompilatorer och interpretatorer / *Compilers and interpreters*
## **TDDB44** Kompilatorkonstruktion / *Compiler construction*

17 dec 2007, 08:00–12:00

**Jour:** 08:00–10:00 Christoph Kessler (visiting 09:30-09:50), 070-3666687, 013-282406
10:00–12:00 Jonas Wallgren, tel. 013-282682

**Hjälpmedel /** *Admitted material:*

– Engelsk ordbok / *Dictionary from English to your native language*;
– Miniräknare / *Pocket calculator*

## General instructions

- This exam has 9 assignments and 5 pages, including this one.
  Read all assignments carefully and completely before you begin.
- The first assignment (on formal languages and automata theory) is ONLY for TDDB29, while the last one (on code generation for RISC...) is ONLY for TDDB44.
- It is recommended that you use a new sheet for each assignment. Number all your sheets, and mark each sheet on top with your name, personnummer, and the course code.
- You may answer in either English or Swedish.
- Write clearly. Unreadable text will be ignored.
- Be precise in your statements. Unprecise formulations may lead to a reduction of points.
- Motivate clearly all statements and reasoning.
- Explain calculations and solution procedures.
- The assignments are *not* ordered according to difficulty.
- The exam is designed for 40 points (per course). You may thus plan about 5 minutes per point.
- Grading: U, 3, 4, 5. For exchange students (with a $P$ in the personnummer) ECTS marks will be applied.

  The preliminary threshold for passing (grade 3) is 20 points.
- **OBS C:are antagna före 2001:** Om du vill ha ditt betyg i det gamla betygsystemet (U, G, VG) skriv detta tydligt på omslaget av tentan. Annars kommer vi att använda det nya systemet (U, 3, 4, 5).

1. **Only TDDB29:** (6 p.) **Formal languages and automata theory**

   Consider the language $L$ consisting of all strings $w$ over the alphabet $\{0, 1\}$ such that if $w$ contains $00$ (i.e., at least 2 zeroes in sequence) then it must contain an odd number of ones.

   (a) Construct a regular expression for $L$. (1.5p)

   (b) Construct from the regular expression an NFA recognizing $L$. (1.5p)

   (c) Construct a DFA recognizing $L$, either by deriving from the NFA of question (1b), or by constructing one directly. (2.5p)

   (d) Give an example of a formal language that is context-free but cannot be recognized by a finite automaton. (0.5p)

2. (6.5p) **Compiler structure**

   (a) Describe what phases normally are found in a compiler, what is their purpose, how they are connected, and what is their input and output. (3p)

   (b) Most modern compilers have not just one but several intermediate representations.

      i. Explain how these are, in general, related to each other, and what this organization means for the code generation process. (1p)

      ii. What is the main advantage of having more than one IR? (0.5p)

   (c) How does a just-in-time (JIT) compiler work? (1.5p)
   Under what condition is execution with a JIT compiler faster than using an ordinary interpreter? (0.5p)

3. (4.5p) **Top-Down Parsing**

   Given a grammar with nonterminals $\langle Expr \rangle$, $\langle Call \rangle$ and $\langle Params \rangle$ and the following productions:

   $$\langle Expr \rangle \rightarrow \textbf{id} \mid \langle Call \rangle$$
   $$\langle Call \rangle \rightarrow \textbf{id} \; ( \; \langle Params \rangle \; )$$
   $$\langle Params \rangle \rightarrow \langle Params \rangle \; , \; \langle Expr \rangle \mid \langle Expr \rangle$$

   where $\langle Expr \rangle$ is the start symbol.

   What is/are the problem(s) with this grammar if it is to be used for writing a recursive descent parser with a single token lookahead? Resolve the problem(s), and write a recursive descent parser for the modified grammar. *(Pseudocode is fine. Use function* `scan()` *to read the next input token.)* (4.5p)

4. (6 p.) **LR parsing**

   Given the following grammar $G$ for strings over the alphabet $\{x, y, z, t\}$, with nonterminals $A$, $B$ and $C$, where $A$ is the start symbol:

   $$\begin{aligned} A &::= B\ x \mid C \\ B &::= y\ t\ B \mid y \\ C &::= x\ C\ z \mid y \end{aligned}$$

   Is the grammar $G$ in SLR(1)? Is it LR(0)? Motivate with the LR-item sets.

   Construct the characteristic LR-item NFA, the corresponding GOTO graph, the ACTION table and the GOTO table.

   Show with tables and stack how the string $xxyzz$ is parsed.

5. (2 p.) **Memory management**

   What property of programming languages requires the *static link* in the procedure's activation record? What is the purpose of the static link, i.e. how and when is it used? How does the static link differ from the *dynamic link*?

6. (6 p.) **Syntax-directed translation**

   The following grammar rule describes a for loop

   $$\langle loop \rangle \ ::= \ \texttt{for}\ \langle id \rangle\ \texttt{in}\ \langle expr \rangle..\langle expr \rangle\ \texttt{do}\ \langle stmt\_list \rangle\ \texttt{enddo}$$

   where loop variable $\langle id \rangle$ will, at run time, take the values between the two expressions (the endpoints included) one by one, and for each value, $\langle stmt\_list \rangle$ is executed. For example,

   ```
   for i in 2..7 do
      print(i);
   enddo
   ```

   prints the numbers 2 to 7.

   Write a syntax-directed translation scheme, using attributes and semantic rules, for the grammar rule above. The values of the two $\langle expr \rangle$ are computed in the beginning and cannot be changed during the execution of the loop. Neither can the loop variable $\langle id \rangle$ be changed inside the loop.

   You may either use symbolic labels (generated by *newlabel()*) or work directly on quadruple numbers, using backpatching if necessary, but be consistent. Temporary variables can be generated by *gentemp()*.

7. (7 p.) **Intermediate code generation**

   (a) Translate the following code segment into abstract syntax tree, quadruples, and postfix code: (4.5 p)

```
x = 17;
y = 3;
while (x>y) {
  x = x+1;
  if (y<20)
    y = 3*(x-y);
  else
    y = 2*y;
}
print(y);
```

   (b) Given the following program fragment in pseudo-quadruple form:

```
1:  T1 := a + b
2:  y  := T1
3:  T2 := - c
4:  x  := T2 * y
5:  T3 := y > 0
6:  if T3 goto 12
7:  T4 := x < 0
8:  if T4 goto 1
9:  T5 := x + y
10: y := T5;
11: goto 3
12: m := x * y
```

   Divide this program fragment into *basic blocks* and then draw the *basic block graph* for the program fragment. (2.5p)

8. (2 p.) **Target code generation**

   Explain the idea of instruction selection by pattern matching. Why is this technique more powerful than simple macro expansion, and what kind of processor architectures will profit most from this technique? (2p)

9. **Only TDDB44:** (6 p.) **Code generation for RISC ...**

   (a) Explain the main similarity and the main difference between superscalar and VLIW architectures from a compiler's point of view. Which one is harder to generate code for, and why? (1.5p)

   (b) Given the following medium-level intermediate representation of a program fragment (derived from a `while` loop):

```
1:   c = 3
2:   e = 1.0
3:   goto 9
4:   a = c / 2
5:   b = a + e
6:   c = a - b
7:   d = e
8:   e = e * 0.5
9:   f = (e > 0.1)
10:  if f goto 4
11:  d = d / b
```

   Identify the live ranges of program variables, and draw the live range interference graph

   (i) for the loop body in lines 4–9,

   (ii) for the entire fragment.

   For both (i) and (ii), assign registers to all live ranges by coloring the live range interference graph. How many registers do you need at least, and why? (3.5p)

   (c) Register allocation and instruction scheduling are often performed separately (in different phases). Explain the advantages and problems of this separation. (1p)

Good luck!