Linköpings universitet
IDA – Department of Computer and Information Science
Professor Peter Fritzson

TENTAMEN / *EXAM*

TDDD16 Kompilatorer och interpretatorer / *Compilers and Interpreters*
TDDB29 Kompilatorer och interpretatorer / *Compilers and Interpreters*
TDDB44 Kompilatorkonstruktion / *Compiler Construction*

15 April 2009, 08:00–12:00, TDDD16/TDDB29 KÅRA,
TDDB44 TER2

Jour: Kristian Stavåker, 0763-361782, 013-284093; (Will come approx 9.15-9.45 and 10.45-11.15)

Hjälpmedel / *Allowed material:*

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

General Instructions

- This exam has 9 assignments and 4 pages, excluding this one.
- Read all assignments carefully and completely before you begin.
- The first assignment (on formal languages and automata theory) is ONLY for TDDD16/TDDB29, while the last one (on code generation for RISC, etc.) 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, personal number/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 6 minutes per point.
- Grading: U, 3, 4, 5.
- For exchange students (with a in the personnummer) ECTS marks will be applied.
- The preliminary threshold for passing (grade 3) is 20 points.

1. **Only TDDD16/TDDB29:** (6p) **Formal Languages and Automata Theory**

Consider the language *L* consisting of all strings *w* over the alphabet {a,b} such that *w* contains bb or aaa (i.e., at least 2 b:s in sequence, or at least 3 a:s in sequence, or both). For example, the strings ababb, aaaba, bbaaab, bb, etc. belong to the language *L* and the string bab does not belong to the language *L*.

(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 not context-free. (0.5p)

2. (6p) **Compiler Structure and Generators**

(a) What are the advantages and disadvantages of a multi-pass compiler (compared to a one-pass compiler)? (1p)

(b) Describe briefly what phases (at least 4 phases, max 5) normally are found in a compiler, what is their purpose, how they are connected, what is their input and output. (3p)

(c) 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, and what could be a drawback? (1 p)

3. (4p) **Error Handling**

Explain, define, and give examples of using the following concepts regarding error handling:

a) Valid prefix property. (1p)
b) Minimum distance. (1p)
c) Phrase level recovery. (1p)
d) Global correction. (1p)

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

Given the following grammar G for strings over the alphabet {`y,w,z,t`}with nonterminals `A`, `D` and `C` where `A` is the start symbol:

```
A ::= y D y  | C
D ::= w t D  | w
C ::= y C z  | w
```

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 `ywtwy` is parsed.

5. (3 p) **Symbol Table Management**

The C language allows static nesting of scopes for identifiers, determined by blocks enclosed in braces.

Given the following C program:

```
int k;
int main( void )
{
  int i;
  // ... some statements omitted
  if (i==0) {
    int j, k;
    // ... some statements omitted
    for (j=0; j<100; j++) {
      int i;
      // ... some statements omitted
      i = k * 2;
    }
  }
}
```

For the program point containing the assignment `i = k * 2`, show how the program variables are stored in the symbol table if the symbol table is to be realized as a hash table with chaining and block scope control. Assume that your hash function yields value 2 for `i`, value 1 for `j` and `k`, and value 4 for `main`. (2p)

Show and explain how the right entry of the symbol table will be accessed when looking up identifier k in the assignment `i = k * 2`. (0.5p)

When generating code for a block, one needs to allocate run-time space for all variables defined in the block. Given a hash table with chaining and block scope control as above, show how to enumerate all variables defined in the current block, without searching through the entire table. (0.5p)

6. (5 p) Syntax-directed translation

The following grammar rule describes a for loop:

```
<loop> ::= for <id> in <expr> .. <expr> do <stmt_list> enddo
```

where loop variable <id> will, at run time, take the values between the two expressions (the endpoints included) one by one, and for each value, <stmt_list> 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. You will typically need to factorize the grammar, i.e., split this rule into several rules to achieve what you need. The values of the two <expr> are computed in the beginning and cannot be changed during the execution of the loop. Neither can the loop variable <id> 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. (6.5 p) Intermediate Code Generation

Given the following code segment:

```
x = 16;
y = 3;
while (x>y) {
  x = x+1;
```

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

(a) Translate the code segment into abstract syntax trees, quadruples, and postfix code. (4.5 p)

(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. (2p)

8. (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?

9. (1.5 p) **Target code generation**

Explain the idea of instruction selection by pattern matching. Why is this technique more powerful than simple macro expansion?

10. **Only TDDB44:** (6 p) **Code Generation for RISC, etc.**

(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 `for`-loop):

```
1: c = 3;
2: k = 20;
3: if k<=0 goto 9;
4: a = c / 2;
5: b = a + c;
6: c = a * b;
7: k = k - 1;
8: goto 3;
9: d = b * c
```

Identify the live ranges of program variables, and draw the live range interference graph for the entire code fragment.

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!