Linköpings universitet
IDA Department of Computer and Information Sciences
Prof. Peter Fritzson and Doc. Christoph Kessler

TENTAMEN / *EXAM*

TDDB29 Kompilatorer och interpretatorer / *Compilers and interpreters*

TDDB44 Kompilatorkonstruktion / *Compiler construction*

16 dec 2006, 08:00–12:00

Jour: Christoph Kessler (070-3666687, 013-282406)

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. The preliminary threshold for 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 terminate with a 1.

   (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. (4p) **Phases and passes**

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

   (b) 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)

3. (5p) **Top-Down Parsing**

   (a) Given a grammar with nonterminals S and X and the following productions:
   $S \rightarrow aS \mid aX$
   $X \rightarrow XbX \mid d$
   where S 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)

   (b) We learned that any regular language can also be expressed by a context-free grammar. So, why don't we simply use the parser for lexical analysis, too? (0.5p)

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

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

$$A ::= xAyAz \mid xBzAx \mid \epsilon$$
$$B ::= yBzBx \mid yBxBz \mid \epsilon$$

   If G is SLR(1) or even LR(0), construct the canonical LR-items and the LR-item DFA for the grammar and show with tables and stack how the string `xyz` is parsed. *If G* is *not* SLR(1) or LR(0), then explain why.

5. (3 p.) **Memory management**

   What is an activation record? What properties of a programming language lead to a need for activation records? What does an activation record contain?

6. (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)

7. (6 p.) **Syntax-directed translation**

   The REPEAT statement in a Pascal-like language could be described using this rule:

```
<rep-stmt> ::= REPEAT <stmt> UNTIL <expr>
```

   The semantics of the REPEAT statement is that statement `<stmt>` is executed and then repeated as long as expression `<expr>` evaluates to zero.

   Write the semantic rules — a syntax directed translation scheme — for translating the REPEAT statement to quadruples. Assume that the translation scheme is to be used in a bottom-up parsing environment using a semantic stack. Use the grammar rule above as a starting point, but maybe it has to be changed.

   You are not allowed to define and use symbolic labels, i.e., all jumps should have absolute quadruple addresses as their destinations. Explain all the attributes, functions, and instructions that you introduce. State all your assumptions.

8. (7 p.) **Intermediate code generation**

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

```
x = 123;
y = 3;
if (x>100) {
  x = x - y;
  y = 2*y;
}
else
  y = 2*x;
foo(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 13
7:  T4 := x < 0
8:  if T4 goto 1
10: T5 := x + y
11: y := T5;
12: goto 3
13: m := x * y
```

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

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

   (a) What is branch prediction, and when is it used? Give an example! Why is it important for pipelined processors? (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 8
4:   a = c / 2
5:   b = a + e
6:   c = a * b
7:   e = e / 2
8:   f = (e > 0.1)
9:   if f goto 4
10:  d = 1 / c
```

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

(i) for the loop body in lines 4–8,

(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!