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*

19 Augusti 2009, 14:00–18:00, TER2

Jour: Kristian Stavåker, 0763-361782, 013-284093; (Will come approx 15.00 and 16.30)

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, including 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 {c,d} such that *w* contains dd or ccccc (i.e., at least 2 d:s in sequence, or at least 5 c:s in sequence, or both). For example, the strings cdcdd, ccccdc, ddccccccd, dd, etc. 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 correction. (1p)
d) Global correction. (1p)

4. (6 p) LR Parsing

Given the following grammar G for strings over the alphabet {`a,+,*,(,)`}with nonterminals `E`, `T`, and `F` where `E` is the start symbol:

```
E::=T|E+T
T::=F|T*F
F::=a|(E)
```

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:

```
a*a+a*(a+a)
```

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

A Pascal-like language is extended with a `restartblock` statement according to the following grammar:

```
<block>     ::= begin <stmt_list> end
<stmt_list> ::= <stmt_list> <stmt> |
<stmt>      ::= <assignment> | ... | restartblock
```

(where " . . . " represents all other possible kinds of statements).

`restartblock` means that execution restarts at the beginning of the immediately enclosing block.

Example:

```
begin
i:=7;
L1: begin
  j:=j+1;
  if j<i
  L2: then restartblock
  else i:=i+1
  end;
end;
```

`restartblock` at L2 therefore means a jump to L1 (i.e., the beginning of the enclosing block).

a) Write a syntax-directed translation scheme, with attributes and semantic rules, for the above grammar section.

b) What problem would occur in the handling of the translation scheme if instead of `restartblock` there would be an `exitblock` that jumped to the end of the immediate enclosing block (instead of begin).

7. (8 p) Intermediate Code Generation and Optimization

Given the following code segment:

```
for i:=1 to 20 do
if i>15
then x:=x+1
else y:=y-1;
```

(a) Translate the code segment into abstract syntax trees, quadruples, and postfix code. (3 p)

(b) Divide the following code segment into basic blocks, draw a control flow graph, and show as well as motivate the existing loops: (3 p)

```
goto L2
L1: x:=x+1
L2: x:=x+1
L3: x:=x+1
if x=1 then goto L1
if x=2 then goto L3
if x=3 then goto L5
L4: x:=x+1
L5: x:=x+1
if x=4 then goto L4
```

(d) Explain, using clear and well defined examples, the handling of induction variables and loop invariants at loop optimization. (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. 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) What is branch prediction and when is it used? Give an example! Why is this important for pipelined processors? (2.5p)

(c) Explain briefly the concept of software pipelining. Show it with a simple example. (1p)

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

Good luck!