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*

17 December 2009, 8:00–12:00, T1 and TER1

Jour: Peter Fritzson, 0708-281484 and 013-281484; (Will come approx 9.00 and 10.30)

Hjälpmedel / *Allowed material:*

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

General Instructions

- This exam has 11 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 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,h} such that *w* contains hhh or cccc (i.e., at least 3 h:s in sequence, or at least 4 c:s in sequence, or both). For example, the strings chchhh, ccccchc, hhhcccch, hhh, 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. Only TDDB44 (3p) Compiler Lab Exercises

Correct and complete labs from the 2009 TDDB44 lab course handed in at the latest Dec 15, 2009, will give 3 points. State if you think that you have fulfilled these conditions and should receive these points.

3. (3p) Compiler Structure and Generators

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

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

4. (3p) Error Handling

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

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

5. (5p) Top-Down Parsing

(a) Given a grammar with nonterminals `S`, `A`, `B` and the following productions:

```
S ::= S 1 | A B 2 | A B 3

A ::= A 4 | 5

B ::= B 6 | e
```

where `S` is the start symbol, `1, 2, 3, 4, 5 and 6` are terminals. (***e is the empty string!***)

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/program code without declarations is fine. Use the function `scan()` to read the next input token, and the function `error()` to report errors if needed.) (4.5p)

(b) The theory for formal languages and automata says that a stack is required for being able to parse context-free languages. We have used such a stack, for instance, in the LL-item pushdown automaton in the lecture on top-down parsing. But where is the corresponding stack in a recursive descent parser? (0.5p)

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

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

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

is parsed.

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

For the program point containing the assignment `i = m * 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 3 for `i`, value 1 for `j` and `m`, and value 4 for `main`. (2p)

Show and explain how the right entry of the symbol table will be accessed when looking up identifier m in the assignment `i = m * 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)

8. (6 p) **Syntax-directed translation**

The following grammar rules inspired by the Modelica language describes a for loop

```
<loop>  ::= for <id> in <range> loop <statement_list> end for;
<range> ::= <startexpr> : <stepexpr> : <endexpr>
```

where loop variable `<id>` will, at run time, take the values starting with `<startexpr>`, increases by `<stepexpr>` for each iteration, and ending if greater or equal to `<endexpr>`. For each value, `<statement_list>` is executed. For example,

```
for i in 2 : 2 : 9 loop
  print(i);
end for;
```

prints the numbers 2, 4, 6, 8.

Write a syntax-directed translation scheme, using attributes and semantic rules, for the grammar rule above. The values of `<startexpr>`, `<stepexpr>` and `<endexpr>` are computed at 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()`.

9. (6 p) Intermediate Code Generation and Optimization

Given the following code segment:

```
for i:=1 to 20 do
if i>a+5
then x:=x+2
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
```

10. (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? What is a display and how is it used)?

11. 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) What is a live range? Explain the concept and show a simple example. (1p)