Tentamen/Exam

TDDB44 Kompilatorkonstruktion / Compiler Construction
TDDD55 Kompilatorer och interpretatorer /
Compilers and Interpreters

2015–04–10, 14.00 – 18.00

Hjälpmedel / Allowed material:

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

General instructions:

- Read all assignments carefully and completely before you begin
- **Note that not every problem is for all courses.** Watch out for comments like "TDDD55 only".
- You may answer in Swedish or in English.
- Write clearly — unreadable text will be ignored. Be precise in your statements — unprecise formulations may lead to 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 6 minutes per point.
- Grading: U, 3, 4, 5 resp. Fx, C, B, A.
- The preliminary threshold for passing (grade 3/C) is 20 points.

1. (TDDD55 only - 6p) **Formal Languages and Automata Theory**
   Consider the language L consisting of all strings w over the alphabet $\{0, 1\}$ containing at least 3 symbols such that the third symbol from the end in w is 0. Example of strings in the language: $11\underline{0}11$, $\underline{0}00$, $010101\underline{0}10$

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

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

   (c) (2.5p) Construct a DFA recognizing L, either by deriving it from the NFA or by constructing it directly.

   (d) (0.5p) Why is $L_1 = \{a^m b^n c^n d^m\}$ a context-free language but not $L_2 = \{a^m b^n c^m d^n\}$? (For all strings in L_1 the number of a:s and d:s are the same, and the number of b:s and c:s are the same. For all strings in L_2 the number of a:s and c:s are the same, and the number of b:s and d:s are the same.)

2. (3p) **Compiler Structure and Generators**

   (a) (2p) What are the generated compiler phases and what are the corresponding formalisms (mention at least 5) when using a compiler generator to generate a compiler?

   (b) (1p) Most modern compilers have not just one, but several intermediate representations(IR). What is the advantage of having more than one IR, and what could be the drawback?

3. (3p) **Symbol Table Management**
   The C language allows static nesting of scopes for identifiers, determined by blocks enclosed in braces. Given the C program below (next page)

   (a) (2p) 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.

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

   (c) (0.5p) After code for a block is generated, one needs to get rid of the information for all variables defined in the block. Given a hash table with chaining and block scope control as above, show how to "forget" all variables defined in the current block, without searching through the entire table.

```
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;
    }
  }
}
```

4. (6p) **Top-Down Parsing**
Given a grammar with nonterminals S, P, Q, and R, where S is the start symbol, and the following productions:

```
1. S ::= P $ P
2. P ::= P £ Q
3.     | Q
4. Q ::= Q # R
5.     | R
6. R ::= 0
7.     | 1
```

Assume that $, £, and # are operators.

(a) (1p) What is the associativity (right, left, none) of the operators?

(b) (1p) What is the precedence (relative priority) between the operators?

(c) (2p) Can the grammar be used directly for a recursive-descent parser? Motivate your answer. If not, rewrite the grammar so that the language it defines can be parsed using the recursiuve-descent method.

(d) (2p) Write a recursive-descent parser to analyze the language defined by the 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.)

5. (TDDD55 only - 6p) **LR parsing**

(a) (3p) Use the SLR(1) tables below to show how the string `a^b*a+b` is parsed. You should show, step by step, how stack, input data etc. are changed during the parsing. Start state is 00, start symbol is `S`.

Grammar:

```
1. S ::= X + X
2. X ::= Y * X
3.     | Y
4. Y ::= Y ^ Z
5.     | Z
6. Z ::= a
7.     | b
```

Tables:

	Action									
State	$	+	*	^	a	b	S	X	Y	Z
00	*	*	*	*	S11	S10	05	04	08	12
01	*	*	*	*	S11	S10	*	06	08	12
02	*	*	*	*	S11	S10	*	07	08	12
03	*	*	*	*	S11	S10	*	*	*	09
04	*	S01	*	*	*	*	*	*	*	*
05	A	*	*	*	*	*	*	*	*	*
06	R1	*	*	*	*	*	*	*	*	*
07	R2	R2	*	*	*	*	*	*	*	*
08	R3	R3	S02	S03	*	*	*	*	*	*
09	R4	R4	R4	R4	*	*	*	*	*	*
10	R7	R7	R7	R7	*	*	*	*	*	*
11	R6	R6	R6	R6	*	*	*	*	*	*
12	R5	R5	R5	R5	*	*	*	*	*	*

(b) (3p) Explain the concept of conflict in LR parsing — what it is, how it could be handled.

6. (TDDB44 only - 6p) **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. (5p) **Syntax-Directed Translation**
An Algol-like language is augmented with an if2-statement in the following way:

```
<if2_statement> ::= if2(<expression_1>,<expression_2>)
                    both: <statement_1>
                    first: <statement_2>
                    secnd: <statement_3>
                    none: <statement_4>
                    endif2;
```

The if2-statement works like the following nesting of if statements:

```
if <expression_1>
   then if <expression_2>
           then <statement_1>
           else <statement_2>
   else if <expression_2>
           then <statement_3>
           else <statement_4>;
```

Write the semantic rules - a syntax directed translation scheme - for translating the if2-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. (6p) **Intermediate Code Generation**

   (a) (3p) Given the following code segment in a Pascal-like language:

   ```
   if x<y and func(func(x))>1
      then repeat
             y:=y+k
           until x<y or y<z
      else x:=13;
   ```

   Translate the code segment into an abtract syntax tree, quadruples, and postfix code.

   (b) (3p) Divide the following code inte basic blocks, draw a control flow graph, and show as well as motivate the existing loop(s).

   ```
   L1: x:=x+1
   L2: y:=y+1
   L3: z:=z+1
       if x=1 then goto L3
       x:=x+1
       if x=2 then goto L4
       goto L2
   L4: x:=x+1
       if x=4 then goto L1
   ```

9. (5p) **Memory management**

   (a) (1p) What does an activation record contain?

   (b) (2p) What happens on the stack at function call and at function return?

   (c) (2p) What are static and dynamic links? How are they used?

10. (TDDB44 only - 6p) **Code Generation for RISC etc.**

   (a) (1p) 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?

   (b) (2p) What is branch prediction and when is it used? Give an example! Why is this important for pipelined processors?

   (c) (3p) Given the following(next page) medium-level intermediate representation of a program fragment:

```
1: a = 1.0
2: b = 1.0
3: c = 3.0
4: e = 2.0
5: goto 9
6: b = a + b
7: a = c / 2.0
8: c = a * e
9: e = e / 2.0
10: f = (e > 0.1)
11: if f goto 6
12: d = 1.5 * a
```

Identify the live ranges of program variables, and draw the live range interference graph for the entire fragment. Assign registers to all live ranges by coloring the live range interference graph. How many registers do you need at least, and why?