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

2017–04–21, 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 $\{0, 1\}$ such that every string contains 01 <u>once</u> *and* 10 <u>once</u>. The occurences may not overlap, i.e. 010 does not count as one 01 and one 10. Example of strings in the language: 111<u>000</u>011, <u>011</u>11<u>000</u>. Examples of strings *not* in the language: 000<u>1000</u>, 111, 00<u>11</u>1.

   (a) (2p) 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.

2. (4p) **Compiler Structure and Generators**

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

   (b) (3p) Describe briefly what phases are found in a compiler. What is their purpose, how are they connected, what is their input and output?

3. (2p) **Symbol Table Management**
   Describe what the compiler — using a symbol table implemented as a hash table with chaining and block scoped control — does in compiling a statically scoped, block structured language when it handles:

   (a) (0.5p) block entry

   (b) (0.5p) block exit

   (c) (0.5p) a variable declaration

   (d) (0.5p) a variable use.

4. (5p) **Top-Down Parsing**

Given a grammar with nonterminals A, B, and C, and S, where S is the start symbol, and the following productions:

```
1. S ::= A ^ A
2. A ::= B * A
3.    | B
4. B ::= B + C
5.    | C
6. C ::= a
7.    | b
```

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 x*y+x/y 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 ::= P + P
2. P ::= Q * P
3.    | Q
4. Q ::= Q / R
5.    | R
6. R ::= x
7.    | y
```

Tables:

```
                Action                    Goto
                ======                    ====
   State   $    *    +    /    x    y    S   P   Q   R
   -----   ------------------------      ------------
    00     *    *    *    *   S09  S10   01  02  05  08
    01     A    *    *    *    *    *    *   *   *   *
    02     *    *   S03   *    *    *    *   *   *   *
    03     *    *    *    *   S09  S10   *   04  05  08
    04     R1   *    *    *    *    *    *   *   *   *
    05     R3  S06  R3   S11   *    *    *   *   *   *
    06     *    *    *    *   S09  S10   *   07  05  08
    07     R2   *    R2    *    *    *    *   *   *   *
    08     R5   R5   R5   R5    *    *    *   *   *   *
    09     R6   R6   R6   R6    *    *    *   *   *   *
    10     R7   R7   R7   R7    *    *    *   *   *   *
    11     *    *    *    *   S09  S10   *   *   *  12
    12     R4   R4   R4   R4    *    *    *   *   *   *
```

(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,c,d} with nonterminals X
   and Y, where X is the start symbol:

   ```
   X ::= pPp | qPq | pQq | qQp | a
   Y ::= pQp | qQq | pPq | qPp | b
   ```

   Is the grammar G in SLR(1) or even LR(0)? Justify your answer using the LR item sets.
   If it is: construct the characteristic LR-items NFA, the corresponding GOTO graph, the
   ACTION table and the GOTO table and show with tables and stack how the string ppqaqqp
   is parsed.
   If it is not: describe where/how the problem occurs.

4

7. (5p) **Syntax-Directed Translation**

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

```
<intervalselect> ::= intervalselect (<expression>,<expression>,<expression>)
                       low: <statement>
                       mid: <statement>
                      high: <statement>
                 endselect
<statement> ::= <assignment> | ... | <intervalselect>
```

(where "..." represents all other possible kinds of statements). First, the `<expression>`s are evaluated. Then, depending on the resulting value of the first expression being less than the value of the second one, between the values of the second and third ones, or larger than the value of the third one the corresponding statement is executed. Only one of the `<statement>` parts is executed. Then execution continues after the `signselect` statement. (You may decide yourself how to handle the borders ($<$ vs. $\leq$) but your choice must be stated in the solution.)

Examples:

```
intervalselect (4711+13-7,2000,4000)
      low: print("Not this one")
      mid: print("Not this one")
     high: print("This one")
 endselect;

intervalselect (42,10,100)
      low: print("Not this one")
      mid: print("This one")
     high: print("Not this one")
 endselect;
```

Write a syntax-directed translation scheme, with attributes and semantic rules, for translating `signselect` statements to quadruples. The translation scheme should be used during bottom-up parsing. You are not allowed to define and use symbolic labels, i.e. all jumps should have absolute quadruple addresses as their destinations. You may need to rewrite the grammar. Explain all the attributes, functions, and instructions that you introduce. State all your assumptions. (Since it is a syntax-directed translation scheme, not an attribute grammar, generation of a quadruple puts it in an array of quadruples and attribute values are "small" values such as single quadruple addresses.)

8. (3p) **Error Handling**
   Explain, define, and give examples of using the following concepts regarding error handling:

   (a) (1p) Valid prefix property,

   (b) (1p) Phrase level recovery,

   (c) (1p) Global correction.

9. (3p) **Memory management**

   (a) (1p) Non-local references: How does a static link work?

   (b) (1p) Non-local references: How does a display work?

   (c) (1p) Dynamic data: How is the actual size and contents of a dynamic array handled?

10. (6p) **Intermediate Code Generation**

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

    ```
    repeat
        y=y-1;
        x=x+1;
        if y<4711
            then z=z+fib(13)
            else z=z+fac(z)
    until x>65536
    ```

    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).

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

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

(a) (2p) Explain the main characteristics of CISC and RISC architectures, and their differences.

(b) (1.5p) 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?

(c) (1.5p) Explain briefly the concept of software pipelining. Show it with a simple example.

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