Examination in courses

TDDA 37/TDDB 44 Compiler Construction

TDDA 28/TDDB 29 Compilers and Interpreters

2001-04-21 08.00 - 12.00

No books, calculators or other aids are allowed.

Maximum = 40 points. 20 points needed for passing.

Teacher on duty: Jonas Wallgren, only on phone: 178594

Problem 1 (4p) Formal languages and automata

NB! Only TDDA 28/TDDB 29 (Compilers and Interpreters) students should solve this problem!

Give a DFA and a regular expression for the language over {0,1} such that if a string contains 00 (two zeroes in succession) it must contain 11.

Problem 2 (4p) Symbol table

Assume having a hash coded symbol table, with names stored in a separate string area. Given following program

```
 PROGRAM Foo1;
 VAR Ida, Ceasar: INTEGER;
    PROCEDURE Foo2;
    VAR Kristina, Ida: BOOLEAN;
        PROCEDURE FOO3;
        VAR Ida, Hejsan: REAL;
(* L1 *)
        BEGIN .... END;
   BEGIN ... END;
(* L2 *)
 BEGIN ... END.
```

show how linked hash table, symbol table (including type information) and block table look at positions L1 and L2 during compilation. You may assume that all names have unique hash values.

Problem 3 (4p) Top-down parsing

When adapting a grammar to recursive-descent parsing two especially important transformations are done.

a) Name and define the transformations. The definitions should be general - not only illustrated by some examples.

b) Which problems are avoided by doing the tranformations? Explain what happens in the parser if they are not done.

Problem 4 (5p) SLR(1) parsing

Given the following grammar:

```
1. <S> ::= a <S> a
2.       | <S> c
3.       | c
```

a) Construct the canonical collection of LR(0) items for the grammar.

b) Use the states and state transitions given by a) for constructing action and goto tables.

c) Show how the string `acca` is parsed.

Problem 5 (6p): Intermediate code generation

```
IF a + b = 0
   THEN REPEAT
           x := x + c;
           y := y - c;
        UNTIL x > y
   ELSE a := - b;
```

a) Translate the statement above to reverse Polish notation.

b) Translate the statement above to quadruples.

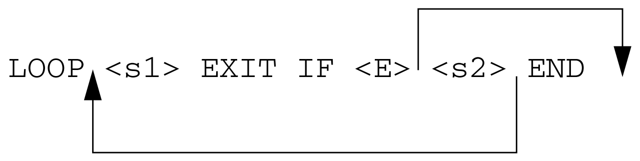
c) Translate the statement above to an abstract syntax tree.

Problem 6 (5p) Syntax directed translation

Ada contains a `LOOP`-construction defined by the rule:

```
<loop-stm>::= LOOP
                   <stm>
                   EXIT IF <expr>
                   <stm>
              END
```

The function of the `LOOP` statement could be illustrated by this picture:



This shows that the statements `<s1>` are executed at least once and `<s2>` (and `<s1>`) are executed when the expression `<E>` is false.

Write the semantic rules - a syntax directed translation scheme - for translating the `LOOP` 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 it has to be changed. `<stm>` and `<expr>` are non-terminals for which you don't need to generate quadruples. Assume that the result of `<expr>` is available in the `<expr>.ADDR` attribute.

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.

Problem 7 (4p): Code optimization

```
 1: T1 := a + b
 2: T2 := T1 - c
 3: x  := T2
 4: T3 := x > 0
 5: if T3 goto 10
 6: T4 := x + 1
 7: x  := T4
 8: T5 := a + b
 9: goto 4
10: m := T5
```

a) Divide the program fragment above into "*basic blocks*" and then draw the control flow graph for the program fragment.

b) Describe the loop optimization methods presented in the course, Use code examples.

Problem 8 (4p) Memory management

a) Explain the concepts of ***display*** and ***static link***. When are they used?

b) What does ***heap allocation*** mean and when is it needed?

c) Explain the concepts of ***fragmentation*** and ***garbage collection***.

d) What is meant by ***static*** and ***dynamic memory allocation***?

Problem 9 (4p) Bootstrapping and compiler generation

Assume there is a new, smashing language X that we want a compiler for. In X itself it's possible to write a compiler that generates good code. Available is a compiler for the language P on the machine M. P isn't powerful enough for writing a compiler for X that generates good code. Show how to construct a compiler running on machine M generating good code for another machine N. Describe the different steps needed and the order of them. Use T diagrams.

Problem 10 (4p): Code generation for RISC

<u>NB! Only TDDA 37/TDDB 44 (Compiler Construction) students should solve this problem!</u>

a) What is branch prediction and when is it used? Give some example! Why is it important for pipelined processors?

b) Explain software pipelining. Give a small example.