Compiler theory

Thursday, June 9, 14:10 - 18:30
Teacher: Daniel Hedin, 021-107052 (15:00 - 17:00)
Allowed aids: Any books, lecture notes and written notes

The exam consists of 50 points distributed over 15 questions. Answers must be given in English or Swedish and should be clearly justified.

- Explain all solutions. A correctly explained solution with minor mistakes may render full points.
- Write clearly. Unreadable solutions will not be graded.
- Start each question on a new page and only write on one side of the page.
- Write down any assumptions you make.
- NOTE: each question can contain more than one part that needs an answer. Read carefully, and make sure you answer everything!

Lexical analysis - regular expressions (4p)

1) Floating point numbers are built by an optional sign followed by digits separated by a decimal point and end with an optional exponent part. Examples of floating point numbers are

```
1.35e+10  +2.54  -5.2  245.3E-2
```

Write a regular expression that recognizes floating point numbers. (2p)

2) Give an example of a language that is not regular and explain why it is not. (2p)

Grammars (6p)

3) Can a left-recursive grammar be LL(1)? Give a left-recursive grammar that is LL(1) or explain why it is impossible. (3p)

4) Assuming that S is the syntactic category of statements and that you would like P to be sequences of statements, what problems can you identify with the following grammar? Give a better definition of sequences of statements. (3p)

$$P \rightarrow S \mid P;P$$

Derivation trees and abstract syntax (8p)

5) Why does the existence of two left-most derivations of one string imply that a grammar is ambiguous? (3p)

6) Write a derivation tree of `3 + 4 * x` given the following grammar. Is it unique? You must justify your answer to get full points. (3p)

$$E \rightarrow VAR \mid NUM \mid E+E \mid E*E$$

7) What is the difference between a derivation tree and an abstract syntax tree? (2p)

Parsing (10p)

8) Write pseudo code for a recursive descent parser for the following language. (7p)

$$\begin{aligned} E &\rightarrow T\ U \\ T &\rightarrow Ta \mid \lambda \\ U &\rightarrow a \mid a\ b \end{aligned}$$

9) In the context of LR parsing, why can shift/reduce conflicts sometimes be tolerated? Does the same hold for reduce/reduce conflicts? (3p)

Type Checking (10p) Consider a simple language of assignments and the corresponding type language

$$\begin{array}{lcl} s & \rightarrow & x = e \mid \tau\, x \mid s_1;s_2 \\ e & \rightarrow & n \mid b \mid x \\ \tau & \rightarrow & int \mid bool \end{array}$$

where x denotes variable names (identifiers), n denotes integers and b denotes booleans. Given the following pseudo code for a type system for the language

```
check(tenv, x = e):
  t1 = tenv[x]; t2 = check e; if (t1 != t2) error

check(tenv, s1 ; s2):
  check(tenv, s1); check(tenv, s2)

check(tenv, t x):
  if (x defined in tenv) error; tenv[x] = t

check(tenv, n): return int
check(tenv, b): return bool
check(tenv, x):
  if (x not defined in tenv) error;
  return tenv[x]
```

10) Show that `int x; bool y; x = 5; y = true` is type correct. (2p)

11) Consider the following extension with code blocks

$$\begin{array}{lcl} s & \rightarrow & x = e \mid \tau\, x \mid s_1;s_2 \mid \{s\} \\ e & \rightarrow & n \mid b \mid x \\ \tau & \rightarrow & int \mid bool \end{array}$$

Modify the pseudo code to handle this extension in such a way that, e.g.,

```
int x; { bool x; x = true }; x = 5
```

and

```
int x; { x = 5 }
```

are type correct, but

```
{ bool y }; y = true
```

is not. (6p)

12) What benefits can you see with static type checking? Discuss from the perspective of the programmer and the compiler implementor. (2p)

Code generation. (12p)

13) Consider the following small language of integer expressions.

$$
\begin{array}{lcl}
s & \to & x = e \mid \mathit{int}\ x \mid s_1 ; s_2 \\
e & \to & n \mid x \mid e + e \mid e * e
\end{array}
$$

Write pseudo code for a code generator that takes a program in the above language and produces Trac-42 stack code. You may assume that all programs are type correct, i.e., that all variables have been declared before being used. (6p)

14) Generate Trac-42 code for `int x; int y; y = 5; x = 15 + 10 * y`. (3p)

15) The following program contains duplicated computation. Rewrite it to be more efficient. (3p)

```
[f]
  LINK
  DECL 1
    LVAL -1(FP)
      PUSHINT 15
      RVALINT 2(FP)
    MULT
  ASSINT
  DECL 1
    LVAL -2(FP)
      PUSHINT 3
        PUSHINT 15
        RVALINT 2(FP)
      MULT
    ADD
  ASSINT
  LVAL 3(FP)
    RVALINT -1(FP)
    RVALINT -2(FP)
  ADD
  ASSINT
  UNLINK
  RTS
```