Mälardalen University
School of innovation, design and engineering

Compiler theory

Thursday, January 14, 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 54 points distributed over 12 questions. Answers must be given in English 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.

Lexical analysis - regular expressions (6p)

1) Give a regular expression that matches C-like multi-line comments. The expression should allow for free use of `*` and `/` inside the comment. (4p)

```
/* ok comment */

/** also
 *  ok comment
 **/
```

2) Explain why it is not possible to create a regular expression that matches nested comments that begin and end with special symbols (`/*` and `*/` for C-like multi-line comments). (2p)

Grammars (12p)

3) As you recall we have talked about two different types of conflicts with respect to LL(1) grammars: FIRST/FIRST conflicts and FIRST/FOLLOW conflicts. The grammar in Figure 1 is not LL(1). Identify all conflicts in the grammar and state their types. (4p)

$$\begin{array}{lcl} S & \rightarrow & T\ U \\ T & \rightarrow & T\ a \mid \lambda \\ U & \rightarrow & a \mid a\ b \end{array}$$

Figure 1: Non-LL(1) grammar

4) Rewrite the grammar in Figure 1 to be LL(1). Illustrate all steps of the process by identifying the what the rewriting aims to remedy, how this is achieved and the resulting intermediate grammar. Argue for why the final grammar defines the same language as the grammar in Figure 1. (6p)

5) Show that the following grammar is ambiguous. (2p)

$$\begin{array}{lcl} S & \rightarrow & a\ A \mid b\ B \\ A & \rightarrow & a\ A\ A \mid b\ S \mid b \\ B & \rightarrow & b\ B\ B \mid a\ S \mid a \end{array}$$

Derivation trees and abstract syntax (10p)

6) Consider the grammar in Figure 2. Draw the derivation tree for the string `x = 1 + y` (3p)

7) Suggest an abstract syntax for the grammar in Figure 2 and show the derivation trees corresponding to `x = y = 1 + x * 3` and `x = y = (1 + x) * 3` given that they were parsed according to the given grammar and then translated to the abstract syntax. (7p)

$$
\begin{array}{lcl}
E_0 & \rightarrow & E_1 \ ASN \\
ASN & \rightarrow & = \ E_1 \ ASN \mid \lambda \\
E_1 & \rightarrow & E_2 \ ADD \\
ADD & \rightarrow & + \ E_2 \ ADD \mid \lambda \\
E_2 & \rightarrow & E_3 \ MUL \\
MUL & \rightarrow & * \ E_3 \ MUL \mid \lambda \\
E_3 & \rightarrow & ID \mid NUM \mid (E_0)
\end{array}
$$

Figure 2: LL(1) expression grammar

Parsing (10p)

8) Consider the grammar in Figure 3 written using bison compatible syntax. Show that the grammar contains an LR(0) reduce/reduce conflict by constructing the LR(0) state machine for the grammar. Make sure to mark in which state the reduce/reduce conflict occurs. (6p)

9) Explain the reduce/reduce conflict studied in question 8 and suggest how to modify the grammar in Figure 3 to remove it. The expressivity of the language must remain the same. (4p)

```
stmt     : fun_decl | expr ';' ;
fun_decl : ID '(' parm ')' stmt;
parm     : ID ;
expr     : ID | ID '(' expr ')' ;
```

Figure 3: Excerpt from grammar

$$\text{asn}\frac{\Gamma(x) = \tau \quad \Gamma \vdash e : \tau}{\Gamma \vdash x \;=\; e \Rightarrow \Gamma} \qquad \text{seq}\frac{\Gamma_1 \vdash s_1 \Rightarrow \Gamma_2 \quad \Gamma_2 \vdash s_2 \Rightarrow \Gamma_3}{\Gamma_1 \vdash s_1; s_2 \Rightarrow \Gamma_3}$$

$$\text{dec}\frac{x \notin dom(\Gamma)}{\Gamma \vdash \tau\, x \Rightarrow \Gamma[x \mapsto \tau]}$$

$$\text{num}\frac{}{\Gamma \vdash n : int} \qquad \text{bool}\frac{}{\Gamma \vdash b : bool} \qquad \text{var}\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

Figure 4: Type system

Type Checking (10p) Consider the following 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. An example of a valid program is `int x; bool y; x = 5; y = true`.
The type rules for the language are found in Figure 4 in the form of natural deduction style rules and below in terms of pseudocode.

```
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]
```

Now consider an extension of the language with a simplified variant of records, that are limited to one property. Note how we change the left hand side of assignment from simple variables to locations l, that denote variables or properties in records.

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

10) Give type rules in terms of pseudo code or natural deduction style rules for the new constructions and for the changed constructions: locations l, assignment $l \;=\; e$, projection $e\,.\,x$, and record literals, $\{\, x : e \,\}$. (8p)

11) Show that the following program is type correct in the extended type system. (2p)

```
{ f : int } y; y.f = 1
```

Code generation. (6p) Consider the following program compiled to Trac42 VM code.

12) What does the function `f` starting on line 2 do? Give the corresponding source code, explain what it computes, and how it relates to the Trac42 VM code. (6p)

```
0   BSR 30
1   END
2   [f]
3     LINK
4       RVALINT 2(FP)
5       PUSHINT 2
6     LTINT
7     BRF 13
8       LVAL 3(FP)
9       RVALINT 2(FP)
10      ASSINT
11      UNLINK
12      RTS
13    LVAL 3(FP)
14      DECL 1
15        RVALINT 2(FP)
16        PUSHINT 1
17      SUB
18      BSR 2
19      POP 1
20      DECL 1
21        RVALINT 2(FP)
22        PUSHINT 2
23      SUB
24      BSR 2
25      POP 1
26    ADD
27    ASSINT
28    UNLINK
29    RTS
30  [main]
31    LINK
32    DECL 1
33      LVAL -1(FP)
34      DECL 1
35      PUSHINT 3
36      BSR 2
37      POP 1
38    ASSINT
39    UNLINK
40    RTS
```