Mälardalen University
School of innovation, design and engineering

Compiler theory, take-home examination

Tuesday, January 12, 2021, 08:00 - 12:30
Contact: Daniel Hedin, 021-107052 (09:00 - 12:00)

OBSERVE! You are *not* allowed to cooperate with, receive help from or give help to other people. Apart from this restriction all other aids are allowed.

Handing in the exam is done via Canvas as *one* pdf, word or txt file before the end of the exam. Exams handed in late will not be graded.

The exam consists of 50 points distributed over 6 questions. Answers must be given in English or Swedish and should be clearly justified.

Grades: 3: 50%, 4: 70%, 5: 90% (corresponding to ECTS E, C and A).

- Explain all solutions. A correctly explained solution with minor mistakes may render full points.
- Start each question on a new page and only write on one side of the page.
- Please make sure your exam is sorted with the answers in the natural order.
- Write down any assumptions you make.

Regular expressions and lexing (4p)

1) Consider the following part of a Gplex specification that writes to the console instead of returning tokens. **(2+2p)**

```
a     { Console.Write("1"); }
ca*   { Console.Write("2"); }
a*c   { Console.Write("3"); }
```

What does the lexer print on input *acaaccac* and why? Explain clearly in terms of the two rules used to disambiguate lexers.

Grammars and parsing (16p)

2) Show how to put the grammar in Table 1 on LL(1) form using llanalyze. **(4+2p)**

$$
\begin{array}{lll}
E & ::= & E \text{ ``.'' } ID \mid \text{``\{''} \; FldsOpt \; \text{``\}''} \mid NUM \mid ID \\
FldsOpt & ::= & \lambda \mid Flds \\
Flds & ::= & ID \text{ ``:'' } E \mid Flds \text{ ``,'' } Flds
\end{array}
$$

Table 1: Simple grammar

Where NUM represents numbers, ID represents identifiers, and terminals are marked with "". You should explain every step you take and why. For full points you should hand in a grammar file that passes llanalyze.

A note on the language of the grammar: the language is a small expression language with variables and structures and should be understood as part of a larger language where variables are declared, e.g., the language of assignment 2.

3) Extend the grammar in Table 1 with assignments to variables and fields, i.e., to be able to write programs of the form below (one example per line) and suggest an abstract syntax for the extended grammar in C# as a collection of classes. (Note: there are two parts in this question: the extension and the abstract syntax in C#.) **(4+6p)**

```
x = 1
x = { a : 15 }
x.a = 12
x.b = { c : 42 }
y = { a : a.b =  { c : 45 }}
x.a = y.b
x.y.z = 15
{ a : 15 }.a
```

A note on the extension: the grammar does not have to be in LL(1) form.

Type checking (10p)

4) Given the type language **(10p)**

$$
\begin{array}{lll}
T & ::= & \texttt{num} \mid \text{"\{"}\ TFldsOpt\ \text{"\}"} \\
TFldsOpt & ::= & \lambda \mid TFlds \\
TFlds & ::= & ID\ \text{":"}\ T \mid TFlds\ \text{","}\ TFlds
\end{array}
$$

that defines the types of the language found in Table 1 (and your extension) implement a type system using the abstract syntax you made in question 3). You can either use pseudocode or C#. Use your intuition for a language with structures and the examples given below.

```
{ } has type { }
{ x : 12 } has type { x : num }
{ a : 42, b : { x : 14 } } has type { a : num, b : { x : num } }
x = 12 has type num  if x has type num
x.a = { b : 15 } if x has type { a : { b : num}, ... }
{ a : t }.a has type t for any type t
x.a has type t if x has type { a : t, ... }
```

A note on the type system: since the language does not have variable declaration you can assume the existence of a variable type environment mapping variable names to the corresponding types.

Code generation and Evaluation. (20p)

5) Implement an evaluator using the abstract syntax you made in question 3). You can use either pseudocode or C#. Use your intuition for a language with structures and the examples given below. You can assume that the programs to be evaluated are type correct. **(10p)**

6) Reverse compile the following Trac42 program, show the correspondence between the resulting program and the original Trac42 code (mark this clearly!) and optimize the source program as much as possible using constant folding and constant propagation. Your answer should contain both the unoptimized result of the program and the optimized version. **(6+4p)**

```
0   BSR 40
1   END
2   [f]
3     LINK
4     DECL 1
5     DECL 1
6     DECL 1
7       LVAL -1(FP)
8         PUSHINT 100
9         PUSHINT 2
10      DIV
11    ASSINT
12      RVALINT -1(FP)
13      PUSHINT 10
14    EQBOOL
15    BRF 19
16        LVAL -2(FP)
17        RVALINT 2(FP)
18      ASSINT
19      LVAL -2(FP)
20        RVALINT -1(FP)
21        PUSHINT 300
22      ADD
23    ASSINT
24      LVAL -3(FP)
25          RVALINT 2(FP)
26          RVALINT 3(FP)
27        ADD
28        RVALINT -2(FP)
29      ADD
30    ASSINT
31    LVAL 4(FP)
32        RVALINT -3(FP)
33        PUSHINT 10
34      ADD
35      PUSHINT 23
36    ADD
37    ASSINT
38    UNLINK
39    RTS
40  [main]
41    LINK
42    DECL 1
43    PUSHINT 3
44    PUSHINT 10
45    BSR 2
46    POP 2
47    POP 1
```