Programming Language Technology

Exam, 14 January 2019 at 8.30–12.30 in M

Course codes: Chalmers DAT151, GU DIT231. As re-exam, also DAT150 and DIT230.
Exam supervision: Daniel Schoepe (+46 31 772 6166), visits at 9:30 and 11:30.

Grading scale: Max = 60p, VG = 5 = 48p, 4 = 36p, G = 3 = 24p.
Allowed aid: an English dictionary.
Exam review: 1 February 2019 10-11.30 in room EDIT 6128.

Please answer the questions in English.

Question 1 (Grammars): Write a labelled BNF grammar that covers the following kinds of constructs of C (also C++, Java):

- Programs: A list of statements enclosed between "`int main() {`" and "`}`".
- Statements:
  - statements formed from expressions by adding a semicolon `;`
  - single variable declarations, e.g., `int x;`
  - `while` loops
- Expressions:
  - integer literals
  - identifiers
  - function calls, i.e., identifiers applied to a tuple of expressions
  - addition (`+`) and multiplication (`*`)
  - less-than comparison of integer expressions (`<`)
  - assignments
  - parenthesized expressions

  Function calls, multiplication and addition are left-associative, comparison is non-associative, assignment is right-associative. Function calls bind strongest, then multiplication, then addition, then comparison, then assignment.
- Types: `int` and `bool`

Lines starting with `#` are comments. An example program is:

```
#include <stdio.h>
#define printInt(e) printf("%d\n",e)
int main () {
  int i; int n;
  i = n = 0;
  while (i < 5) n = n + (i = i + 1) * i;
  printInt(n);
}
```

You can use the standard BNFC categories `Integer` and `Ident`, list categories via the `terminator` and `separator` pragmas, and the `coercions` pragma.
(10p)

CLARIFICATION: Clarification: blocks are not explicitely mentioned in the list of constructs to be covered, thus, they are not included in this grammar. However, it is not considered a mistake if a solution includes them.

Question 2 (Lexing): Consider the alphabet $\Sigma = \{0, 1\}$ and the language $L \subseteq \Sigma^*$ of words in which the number of consecutive 1s is always even.

1. Give a regular expression for language L.
2. Give a deterministic finite automaton for L with no more than 6 states.

(4p)

CLARIFICATION: The *number of consecutive 1s is always even* could be read to allow a single 1, since then there are *no consecutive* 1s, meaning 0 of them, which is an even number. It would habe been clearer to avoid Latin and write *the number of 1s following each other without a 0 inbetween.*

Any of the two interpretations will be accepted as solution.

Question 3 (LR Parsing): Consider the following labeled BNF-Grammar (written in `bnfc`). The starting non-terminal is `P`.

```
P1.     P ::= P "*" F   ;
P2.     P ::= F         ;

F1.     F ::= F "^" E   ;
F2.     F ::= E         ;

EX.     E ::= "x"       ;
EY.     E ::= "y"       ;
EN.     E ::= "-" E     ;
EP.     E ::= "(" P ")" ;
```

Step by step, trace the shift-reduce parsing of the expression

```
- x ^ - - y * x
```

showing how the stack and the input evolves and which actions are performed. (8p)

Question 4 (Type checking and evaluation):

1. Write syntax-directed typing rules for the *expressions* of Question 1. Alternatively, you can write the type-checker in pseudo code or Haskell. In any case, the environment must be made explicit. (7p)

CLARIFICATION: A function `lookupVar` can be assumed if its behavior is described.

2. Write syntax-directed interpretation rules for the *statements* of Question 1, assuming an interpreter for expressions $\gamma \vdash e \Downarrow \langle v; \gamma' \rangle$. Alternatively, you can write the interpreter in pseudo code or Haskell. In any case, the environment must be made explicit. (5p)

CLARIFICATION: A function `lookupVar` can be assumed if its behavior is described.

Question 5 (Compilation):

1. Write compilation schemes in pseudo code or Haskell for the statement and expressions constructions of Question 1. The compiler should output symbolic JVM instructions (i.e. Jasmin assembler). It is not necessary to remember exactly the names of the instructions—only what arguments they take and how they work. (9p)

CLARIFICATION: A function `lookupVar` can be assumed if its behavior is described.

2. Give the small-step semantics of the JVM instructions you used in the compilation schemes in part 1. Write the semantics in the form

$$i : (P, V, S) \longrightarrow (P', V', S')$$

where (P, V, S) is the program counter, variable store, and stack before execution of instruction i, and (P', V', S') are the respective values after the execution. For adjusting the program counter, you can assume that each instruction has size 1. (7p)

Question 6 (Functional languages):

1. For the expressions of lambda-calculus with integers and integer addition we use the grammar

$$e ::= n \mid e + e \mid x \mid \lambda x \to e \mid e\, e$$

and for simple types $t ::= \mathbb{N} \mid t \to t$. Non-terminal x ranges over variable names and n over integer constants 0, 1, etc.

For the following typing judgements $\Gamma \vdash e : t$, decide whether they are valid or not. Your answer should be just "valid" or "not valid".

(a)	$k : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$	$\vdash 1 + k\,(\lambda x \to x)$	$: \mathbb{N}$
(b)		$\vdash \lambda y \to \lambda h \to (h\,1)\,y$	$: \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$
(c)	$x : \mathbb{N} \to \mathbb{N},\ g : \mathbb{N}$	$\vdash (g + 1)\,x$	$: \mathbb{N}$
(d)	$f : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$	$\vdash (\lambda g \to f\,g)\,(\lambda y \to f\,(\lambda z \to z)\,y)$	$: \mathbb{N} \to \mathbb{N}$
(e)	$f : \mathbb{N} \to \mathbb{N}$	$\vdash \lambda x \to f\,(1 + f\,(f\,x))$	$: \mathbb{N} \to \mathbb{N}$

The usual rules for multiple-choice questions apply: For a correct answer you get 1 *point for a wrong answer* -1 *points. If you choose not to give an answer for a judgement, you get* 0 *points for that judgement. Your final score will be between* 0 *and* 5 *points, a negative sum is rounded up to* 0. (5p)

CLARIFICATION: The original problem (d) had a spurious parenthesis at the end: (d) $f : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}) \vdash (\lambda g \to f\, g)\,(\lambda y \to f\,(\lambda z \to z)\,y))) : \mathbb{N} \to \mathbb{N}$

2. Write a **call-by-value** interpreter for above lambda-calculus either with inference rules, or in pseudo code or Haskell. (5p)