Programming Language Technology

Exam, 08 January 2018 at 8.30–12.30 in J

Course codes: Chalmers DAT150/151, GU DIT231. As re-exam, also TIN321 and DIT229/230.
Exam supervision: Andreas Abel (+46 31 772 1731), 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: Tuesday 16 January 2017 at 10.30-12 in room EDIT 6128.

Please answer the questions in English. Questions requiring answers in code can be answered in any of: C, C++, Haskell, Java, or precise pseudocode.

For any of the six questions, an answer of roughly one page should be enough.

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

- statements:
  - blocks: lists of statements (possibly empty) in curly brackets { }
  - variable initialization statement: a type followed by an identifier, the equals sign, an initializing expression, and a semicolon, e.g. `int x = 4;`
  - return statements: an expression between keyword `return` and a semicolon, e.g. `return 1;`
- types: `int`
- expressions:
  - integer literals
  - subtraction `-`
  - multiplication `*`
  - pre-increment of variables `++x`
  - function calls `x(e,...,e)` with zero or more arguments
  - parenthesized expressions `(e)`.

  Both arithmetic operations are left associative. Multiplication binds stronger than subtraction.

An example statement is:

```
{ int x = f (0, 1); return ++x - (2 - 3) - 4 * g();
  int y = 5;        return ++y; }
```

You can use the standard BNFC categories `Integer` and `Ident`. **Do not** use list categories or `terminator/separator` rules. **Do not** use `coercions` either. (10p)

Question 2 (Lexing): Consider the alphabet $\Sigma = \{0, 1\}$ and the language L of bit-streams that contains the sequence 01 *twice*.

1. Give a regular expression for language L.
2. Give a non-deterministic finite automation for L.
3. Give a deterministic finite automaton for L.

(8p)

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

```
S1.     S ::= S ";" C   ;
S2.     S ::= C         ;

C1.     C ::= C "," A   ;
C2.     C ::= A         ;

AA.     A ::= "alice"   ;
AB.     A ::= "bob"     ;
AC.     A ::= "chris"   ;
```

Step by step, trace the LR-parsing of the expression

```
alice ; bob , chris
```

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

Question 4 (Type checking and evaluation):

1. Write syntax-directed *type checking* rules for the *expression* forms of Question 1. The typing environment must be made explicit. (6p)
2. Write syntax-directed *interpretation* rules for the *statement* forms and lists of Question 1. The environment must be made explicit, as well as all possible side effects. (6p)

Question 5 (Compilation):

1. Write compilation schemes in pseudo-code for each of the grammar constructions in Question 1 generating JVM (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. (8p)
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.

You can omit instructions for function call and return. (4p)

Question 6 (Functional languages):

1. For lambda-calculus expressions we use the grammar

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

and for simple types $t ::= \mathtt{int} \mid t \to t$. 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) $\vdash \lambda x \to \lambda y \to (f\, x)\, 0 : \mathsf{int} \to (\mathsf{int} \to \mathsf{int})$.

(b) $g : (\mathsf{int} \to \mathsf{int}) \to \mathsf{int} \vdash g\, (\lambda x \to x) : \mathsf{int}$.

(c) $f : \mathsf{int} \to \mathsf{int} \vdash \lambda x \to f\, (f\, (f\, x)) : \mathsf{int} \to \mathsf{int}$.

(d) $x : \mathsf{int} \to \mathsf{int}, g : \mathsf{int} \vdash g\, x : \mathsf{int}$.

(e) $f : (\mathsf{int} \to \mathsf{int}) \to (\mathsf{int} \to \mathsf{int}) \vdash (\lambda x \to f\, (x\, x))\, (\lambda x \to f\, (x\, x)) : \mathsf{int} \to \mathsf{int}$.

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)

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