Programming Language Technology

Exam, 11 April 2017 at 8.30–12.30 in SB (Sven Hultins gata 6)

Course codes: Chalmers DAT151, GU DIT231. As re-exam, also DAT150, DIT229/230, and TIN321.
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 25 April 2017 at 10-11 in room EDIT 8103.

Please answer the questions in English.

Question 1 (Grammars): Write a labelled BNF grammar that covers the following constructs of a C-like imperative language: A program is a list of statements. Types are `int` and `bool`. Statement constructs are:

- variable declarations (e.g. `int` x`;`), not multiple variables, no initial value
- expression statements (E`;`)
- `while` loops
- blocks: (possibly empty) lists of statements enclosed in braces

Expression constructs are:

- identifiers/variables
- integer literals
- pre-increments of identifiers (`++`x)
- greater-or-equal-than comparisons (E `>=` E')
- assignments of identifiers (x `=` E)

Greater-or-equal is non-associative and binds stronger than assignment. Parentheses around and expression are allowed and have the usual meaning. An example program would be:

```
int x; x = 0; while (10 >= ++x) {}
```

You can use the standard BNFC categories `Integer` and `Ident` as well as list short-hands, and `terminator`, `separator`, and `coercions` rules. (10p)

Question 2 (Lexing): A *string literal* is a character sequence of length ≥ 2 which starts and ends with double quotes `"`. Taking away both the starting and the ending `"`, we obtain a string in which `"` may only appear in the form `""`. Valid string literals are e.g.: `"Hi!"` or `"""Ol"`. Invalid string literals are e.g.: `B"` (does not start with double quotes) `"A` (does not end with double quotes), or `"""` (the middle part `"` is not valid since it is a single `"`).

To simplify matters, we represent character " by a and any other character by b. The valid string literals from above become $abbba$ and $aaabba$ and the invalid ones ba, ab, and aaa. Our alphabet thus becomes $\Sigma = \{a, b\}$.

1. Give a regular expression for string literals (using alphabet Σ). Demonstrate that your regular expression accepts the two valid examples and rejects the three invalid ones. (5p)
2. Give a deterministic or non-deterministic automaton for recognizing string literals (using alphabet Σ). Demonstrate that your automaton accepts the two valid examples and rejects the three invalid ones. (5p)

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

```
Or.     D ::= D "|" C   ;  -- Disjunctions
Conj.   D ::= C         ;

And.    C ::= C "&" A   ;  -- Conjunctions
Atom.   C ::= A         ;

TT.     A ::= "true"    ;  -- Atoms
FF.     A ::= "false"   ;
Var.    A ::= "x"       ;
Parens. A ::= "(" D ")" ;
```

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

```
false | x & true
```

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 *statement* forms and lists of Question 1. The typing environment must be made explicit. You can assume a type-checking judgement for expressions.

   Alternatively, you can write the type-checker in pseudo code or Haskell.

   Please pay attention to scoping details; in particular, the program

   ```
   while (0 >= 0) int x; x = 0;
   ```

   should not pass your type checker! (5p)

2. Write syntax-directed *interpretation* rules for the *expression* forms of Question 1. The environment must be made explicit, as well as all possible side effects.

   Alternatively, you maybe write an interpeter in pseudo code or Haskell. (5p)

Question 5 (Compilation):

1. Write compilation schemes in pseudo code for each of the *expression* 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. (6p)

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) are 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. (6p)

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 $t ::= \texttt{int} \mid t \to t$. 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) $\vdash \lambda x \to \lambda y \to (f\, x)\, y : \mathsf{int} \to (\mathsf{int} \to \mathsf{int})$.

   (b) $y : (\mathsf{int} \to \mathsf{int}) \to \mathsf{int} \vdash y\, (\lambda x \to 1) : \mathsf{int}$.

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

   (d) $y : \mathsf{int} \to \mathsf{int}, f : \mathsf{int} \vdash f\, y : \mathsf{int}$.

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

   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)

Good luck!