Programming Language Technology

Exam, 12 January 2023 at 08.30 – 12.30

Course codes: Chalmers DAT151, GU DIT231.
Exam supervision: Andreas Abel (+46 31 772 1731), visits at 09:30 and 11:30.

Grading scale: Max = 60p, VG = 5 = 48p, 4 = 36p, G = 3 = 24p.
Allowed aid: an English dictionary.
Exam review: 23 January 2023 14.30-15.30 in EDIT meeting room 6128 (6th floor).

Please answer the questions in English.

Question 1 (Grammars): Write a labelled BNF grammar that covers the following kinds of constructs of C/C++ (sublanguage of lab 2):

- Program: a sequence of function definitions.
- Function definition: type followed by identifier, comma-separated parameter list in parentheses, and block.
- Parameter: type followed by identifier, e.g. `int x`.
- Block: a sequence of statements enclosed between `{` and `}`
- Statements:
  - block
  - variable declaration, e.g., `int x;`
  - expression followed by semicolon
  - return statement
  - if-else statement
- Expressions, from highest to lowest precedence:
  - atoms: identifier, integer literal, function call
  - addition (+), left associative
  - less-than comparison (<), non-associative
  - assignment, right associative

  Putting an expression into parentheses makes it an atom.
- Type: `int` or `bool`

Line comments are started by `#`. You can use the standard BNFC categories `Integer` and `Ident` and any of the BNFC pragmas (`coercions`, `terminator`, `separator` ...). An example program is:

```
#include <stdio.h>
#define printInt(x) printf("%d\n",x)
int f (int x, int z) {
  x = (z = 7) + z;
  if (z < x) { int x; printInt(x = z + 2); z = x; } else {}
  return x;
}
int main () { return f(1,2); }
```

(10p)

Question 2 (Lexing): A *floating point literal* is given by the following sequence:

- Optionally: a sign (plus or minus), denoted s.
- A possibly empty sequence of digits (digit shall be d).
- A dot denoted f (for "full stop").
- A non-empty sequence of digits.
- Optionally an exponent, given by the following sequence:
  - Lower or upper case letter "e", denoted e.
  - Optionally: a sign.
  - A non-empty sequence of digits.

So the alphabet is $\Sigma = \{d, e, f, s\}$. Present the language of floating point literals in the following ways:

1. as a regular expression,
2. as a non-deterministic finite automaton (NFA) with no more than 10 states,
3. as a deterministic finite automaton (DFA) with no more than 12 states.

Remember to mark initial and final states appropriately. *Note: since each DFA is a NFA, you can omit the NFA when you are sure that your DFA is correct.* (6p)

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

```
Or.    E ::= C "||" E ;
Conj.  E ::= C        ;
And.   C ::= C "&&" B ;
Bit.   C ::= B        ;
Zero.  B ::= "0"      ;
One.   B ::= "1"      ;
```

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

```
0 && 1 || 1 && 0
```

showing how the stack and the input evolve 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. Functions `lookupVar` and `lookupFun` can be assumed. In any case, the typing environment must be made explicit. (6p)

2. Write syntax-directed interpretation rules for the *statements*, *blocks* and *statement sequences* 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. Functions `evalExp` and `lookupVar` can be assumed. In any case, the environment and control flow must be made explicit. *Take care to handle the return statement correctly!* (7p)

Question 5 (Compilation):

1. Translate the function `f` of the example program of Question 1 to Jasmin. It is not necessary to remember exactly the names of the JVM instructions—only what arguments they take and how they work. Make clear which instructions come from which statement, and determine the stack and local variable limits. (7p)

2. Give the small-step semantics of the JVM instructions you used in the Jasmin code in part 1 (except for return instructions). 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, assume that each instruction has size 1. (6p)

Question 6 (Functional languages):

1. The following grammar describes a tiny simply-typed sub-language of Haskell.

x			identifier
i	$::=$	$0 \mid 1 \mid -1 \mid 2 \mid -2 \mid \ldots$	integer literal
e	$::=$	$i \mid e + e \mid x \mid \lambda x \to e \mid e\, e$	expression
t	$::=$	$\mathsf{Int} \mid t \to t$	type

Application $e_1\ e_2$ is left-associative, the arrow $t_1 \to t_2$ is right-associative. Application binds strongest, then addition, then λ-abstraction.

For the following typing judgements $\Gamma \vdash e : t$, decide whether they are valid or not. Your answer can be just "valid" or "not valid", but you may also provide a justification why some judgement is valid or invalid.

(a) $x : \mathsf{Int} \to \mathsf{Int},\ g : \mathsf{Int} \vdash x\,(g + 1) : \mathsf{Int}$
(b) $k : (\mathsf{Int} \to \mathsf{Int}) \to \mathsf{Int} \vdash (k + 1)\,(\lambda x \to x) : \mathsf{Int}$
(c) $f : (\mathsf{Int} \to \mathsf{Int}) \to (\mathsf{Int} \to \mathsf{Int}) \vdash (\lambda g \to f\,g)\,(\lambda x \to f\,x) : \mathsf{Int} \to \mathsf{Int}$
(d) $f : \mathsf{Int} \to \mathsf{Int} \vdash \lambda x \to f\,(f\,(1 + (f\,x))) : \mathsf{Int} \to \mathsf{Int}$
(e) $\vdash \lambda x \to \lambda y \to (y\,x)\,0 \ : \ \mathsf{Int} \to (\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}) \to \mathsf{Int}$

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. For each of the following terms, decide whether it evaluates more efficiently (in the sense of fewer reductions) in call-by-name or call-by-value. Your answer can be just "call-by-name" or "call-by-value", but you can also add a justification why you think so. *Same rules for multiple choice as in part 1.* (5p)

(a) $(\lambda x \to x + x + x)\ (1 + 2 + 3)$
(b) $(\lambda x \to \lambda y \to y + y)\ (1 + 2 + 3 + 4)\ (5 + 6)$
(c) $(\lambda x \to \lambda y \to y + y)\ (1 + 2)\ (3 + 4 + 5 + 6)$
(d) $(\lambda x \to \lambda y \to y + y)\ ((\lambda z \to z\,z)(\lambda z \to z\,z))\ (3 + 4 + 5 + 6)$
(e) $(\lambda x \to \lambda y \to x + x)\ (1 + 2)\ ((\lambda z \to z\,z)(\lambda z \to z\,z))$

Good luck!