Programming Language Technology

Exam, 16 January 2025, 8.30–12.30 in HA1-4

Course codes: Chalmers DAT151, GU DIT231.
Exam supervision: Andreas Abel (+46 31 772 1731), visits at 9:30 and 11:00.

Grading scale: Max = 60p, VG = 5 = 48p, 4 = 36p, G = 3 = 24p.
Allowed aid: an English dictionary.
Exam review: Wed 29 January 2024 14.30-15.30 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/C++ (sublanguage of lab 2):

- Program: a sequence of function definitions.
- Function definition: type `bool` followed by identifier, comma-separated parameter list in parentheses, and block.
- Parameter: type `bool` followed by identifier, e.g. `bool x`.
- Block: a sequence of statements enclosed between `{` and `}`
- Statements:
  - block
  - initializing variable declaration, e.g., `bool x = true;`
  - expression followed by semicolon
  - `return` statement
  - `while` statement
- Expressions, from highest to lowest precedence:
  - atoms: identifier, boolean literal (`true` or `false`), function call
  - conjunction (`&&`), left associative
  - disjunction (`||`), left associative
  - assignment, right associative

  Wrapping an expression in parentheses makes it an atom.

Line comments are started by `#`. You can use the standard BNFC category `Ident` and any of the BNFC pragmas (`coercions`, `terminator`, `separator` ...). Example program:

```
#include <stdio.h>
#define printBool(x) printf("%d\n",x)
#define bool int
bool f (bool y, bool z) {
  y = z || y;
  while (y && z) { bool y = true; printBool (y = z); z = false; }
  return y;
}
bool main () { return f (false, true); }
```

(10p)

Question 2 (Lexing): Consider the alphabet $\Sigma = \{a, b\}$ and the language $L = \{waa, wab \mid w \in \Sigma^*\}$ of words that end in aa or ab.

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

(6p)

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

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

T1.     T ::= T "&" A    ;
T2.     T ::= A          ;

AX.     A ::= "x"        ;
AY.     A ::= "y"        ;
AZ.     A ::= "z"        ;
```

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

```
x&y;z
```

showing how the stack and the input evolves and which actions are performed. (6p)

Question 4 (Type checking and evaluation):

1. Write syntax-directed *type checking* rules for the *statement* forms and blocks of Question 1. The form of the typing judgements should be $\Gamma \vdash_t s \Rightarrow \Gamma'$ where s is a statement or list of statements, t the return type, Γ is the typing context before s, and Γ' the typing context after s. Observe the scoping rules for variables! You can assume a type-checking judgement $\Gamma \vdash e : t$ for expressions e.

   Alternatively, you can write the type checker in pseudo code or Haskell (then assume `checkExpr` to be defined). In any case, the typing environment and the return type must be made explicit. (6p)

2. Write syntax-directed *interpretation* rules for the *expressions* of Question 1. The form of the evaluation judgement should be $\gamma \vdash e \Downarrow \langle v; \gamma' \rangle$ where e denotes the expression to be evaluated in environment γ and the pair $\langle v; \gamma' \rangle$ denotes the resulting value and updated environment. You can assume a judgement $\gamma \vdash ss \Downarrow v$ stating that statements ss return value v in environment γ.

   Alternatively, you can write the interpreter in pseudo code or Haskell (then assume a function `evalStms` to be defined). A function `lookupVar` can be assumed if its behavior is described. In any case, the environment must be made explicit. (8p)

Question 5 (Compilation):

1. *Statement by statement*, translate the function `f` of the example program of Question 1 to Jasmin. Do not optimize the program before translation!

   To translate the call to `printBool`, assume a Java class `Runtime` with a method `void printBool(boolean)`.

   It is not necessary to remember exactly the names of the JVM instructions—only what arguments they take and how they work. But note that machines like JVM do not have instructions for boolean operators (like `&&` and `||`), thus, you have to use conditional jumps.

   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. (7p)

Question 6 (Functional languages):

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

x			identifier
n	$::=$	$0 \mid 1 \mid -1 \mid 2 \mid -2 \mid \ldots$	numeral
e	$::=$	$n \mid e + e \mid x \mid \lambda x \to e \mid e\,e$	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 invalid.

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

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) $(\lambda x \to x + x)\ ((\lambda y \to \lambda z \to y + y)\ (1 + 2)\ (3 + 4 + 5))$
(b) $(\lambda x \to \lambda y \to x + x)\ (1 + 2 + 3)\ ((\lambda z \to z\,z)(\lambda z \to z\,z))$
(c) $(\lambda x \to \lambda y \to y + y)\ (\lambda z \to (\lambda u \to u\,u)(\lambda u \to u\,u))\ (1 + 2)$
(d) $(\lambda x \to \lambda y \to x + x)\ (1 + 2 + 3)\ (4 + 5 + 6 + 7)$
(e) $(\lambda x \to \lambda y \to x + x)\ (1 + 2)\ (3 + 4 + 5)$