Programming Languages / Programming Language Technology

Exam, 11 March 2010

14.00–18.00 in M

Course codes: Chalmers TIN321, GU DIT229 (BSc); Chalmers DAT150, GU DIT230 (MSc)

Questions: Arnar Birgisson (tel. 5402). Course Responsible: Aarne Ranta (`aarne@chalmers.se`)

Grading scale: Max = 60p, VG = 5 = 48p, 4 = 36p, G = 3 = 24p.

Aids: an English dictionary.

Instructions

This exam has three groups of questions, one **easy**, one **intermediate**, and one **advanced**. Points are distributed in such a way that doing the easy questions is enough to pass the exam (mark 3 or G). All the easy and around half of the intermediate are needed to get mark 4. All the intermediate, or half of them and the advanced, are needed to get the highest mark (5 or VG).

You can get points for the more difficult parts without doing the easier ones.

The bonus points from exercises are added to the total score by the teachers.

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

Text in the answers can be in any of: Bulgarian, Danish, Dutch, English, Estonian, Finnish, French, German, Icelandic, Italian, Norwegian, Polish, Romanian, Russian, Spanish, and Swedish.

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

Group 1: easy questions

1. Write a BNF grammar that covers the following constructs in C++:

- statements: `while` statements, block statements (lists of statements included in curly brackets), and expressions used as statements
- expressions: identifiers, integers, assignments (`x = e`), additions (`a + b`) and less-than comparisons (`a < b`)

An example statement is shown in Question 2. The expressions have their usual precedences. You can use the standard BNFC categories `Integer` and `Ident`, as well as the `coercions` macro. (10p)

2. Show the parse tree and the abstract syntax tree of the statement

```
while (x < 123) {y = x ; x = x + 1 ;}
```

in the grammar that you wrote in Question 1. (5p for each tree)

3. Write typing rules or syntax-directed type checking code (or pseudocode) for `while` statements, block statements, and less-than comparisons (<). Don't forget that block statements open a new scope. (5p)

Write big-step operational semantic rules or syntax-directed interpretation code (or pseudocode) for `while` statements, block statements, and less-than comparisons (<). Don't forget that block statements open a new scope, and that expressions can have side effects. (5p)

Group 2: intermediate questions

4. Write two grammars that recognize sequences of identifiers (e.g. `x y foo x`, including the empty sequence). One grammar should be left-recursive, the other right-recursive. Don't use BNFC's list categories, but just plain BNF rules. You may use BNFC's `Ident` category. (5p)

Trace the LR parsing process of the sequence `x y foo x` with both of your grammars. (3p) Estimate the need of stack memory as a function of sequence length for both grammars. (2p)

5. Write the compilation schemes for all constructs in Question 1 in JVM (using Jasmin assembler). It is not necessary to remember exactly the names of the instructions - only what arguments they take and how they work. (6p)

Show the JVM/Jasmin code for the statement in Question 2 *as generated by your compilation schemes*. (4p)

Group 3: advanced questions

6. Write the big-step operational semantic rules for call-by-value and call-by-name lambda calculus, covering abstractions and applications. (6p)

Give an example of an expression and its evaluation whose termination depends on the chosen strategy. Show a few steps of evaluation with both strategies. (4p).