Programming Language Technology

Re-exam, 21 August 2013 at 14:00 in V

Course codes: Chalmers DAT151, GU DIT230. As re-exam, also TIN321 and DIT229.
Teacher: Aarne Ranta (tel. 1082)
Grading scale: Max = 60p, VG = 5 = 48p, 4 = 36p, G = 3 = 24p.
Aids: an English dictionary.
Exam review: upon individual agreement with the teacher.

Instructions

This exam has two groups of questions, one **easy** 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). From another perspective, the easy questions can be answered by anyone who has managed to do the labs without any further reading. The advanced questions may also require material from the lecture notes and exercises. You can get points for the advanced questions without doing the easier ones.

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: Danish, Dutch, English, Estonian, Finnish, French, German, Icelandic, Italian, Norwegian, Spanish, and Swedish.

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

Group 1: easy questions

1. Write a BNF grammar that covers the following constructs in a C-like imperative language:

- statements:
  - `while` statements (`while (condition) body`)
  - blocks: lists of statements (possibly empty) in curly brackets { }
  - expression statements (`expression ;`)
- expressions:
  - variables
  - function calls (with zero or more arguments)

An example statement is shown in question 2. You can use the standard BNFC categories `Double`, `Integer`, and `Ident` as well as list categories and `terminator` and `separator` rules. (10p)

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

```
while (small(distance(x,y))) {add() ;}
```

in the grammar that you wrote in question 1. (10p)

3. Write syntax-directed type checking rules for `while` statements, expression statements, and function calls. (5p)

Write syntax-directed interpretation rules for `while` statements, expression statements, and function calls. The environment must be made explicit, as well as all possible side effects. (5p)

Group 2: advanced questions

4. Write a regular expression that defines exactly those sequences of 0's and 1's that represent positive integers divisible by 4 (3p). Show an automaton that recognizes these sequences (2p if it is nondeterministic, 7p if it is deterministic)

5. Write compilation schemes 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. (6p)

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

6. Show the operational semantic rules for function application in a functional language, with two different evaluation strategies: call by value and call by name. (6p)

Show two examples of expressions: one having a faster computation with call by value, another with call by name. For both examples, trace the computations with a few steps with each strategy, to show the speed difference. (4p)