Programming Language Technology

Exam, 8 March 2012 at 14–18 in V

Course codes: Chalmers DAT150, 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: Tuesday 3 April at 10:00-12:00 in office 6106, EDIT Building.

Instructions

This exam has two groups of questions, one **easy** and **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.

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: Danish, Dutch, English, Estonian, Finnish, French, German, Icelandic, Italian, Norwegian, Romanian, 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 kinds of constructs in C++:
   - statements:
     - `if` statements with `else`
     - blocks: lists of statements (possibly empty) in curly brackets { }
     - expression statements (`exp ;`)
   - expressions:
     - variables
     - integer literals
     - 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

```
if (debug()) prints(3,x) ; else {}
```

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

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

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

Group 2: advanced questions

4. Trace the LR-parsing of the statement given in Question 2, showing how the stack and the input evolves and which actions are performed. Be careful with lists, so that the actions match your grammar in Question 1. (10p)

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. A Church numeral n is a function that applies an arbitrary function to an arbitrary argument n times. Define, in pure lambda calculus,
   - the Church numerals 0,1,2,3
   - the addition of Church numerals (PLUS)
   - the multiplication of Church numerals (MULT)
   - the exponentiation of Church numerals (EXP), where EXP m n multiplies m with itself n times.

The EXP function is not in the lecture notes, but you can figure it out by following the same pattern as PLUS and MULT. (10p)