Programming Language Technology

Re-exam, 15 January 2014 at 8:30–12:30 in M

Course codes: Chalmers DAT151, GU DIT230. As re-exam, also DAT150.
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 **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, 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 statement in Question 2 by using the standard syntactic constructions of C/C++/Java. A standard solution (following the book and the laborations) needs four forms of statements and three forms of expressions, plus some auxiliary rules for dealing with lists of arguments and statements. You can use the standard BNFC categories `Double`, `Integer`, and `Ident` as well as list categories and `terminator` and `separator` rules. (10p)

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

```
if (debug) if (fail()) print(1) ; else {}
```

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

3. Write syntax-directed type checking rules for `if` statements without `else`, for expression statements, and for function calls. (5p)

Write syntax-directed interpretation rules for `if` statements without `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. Also clearly indicate if there are any conflicts and how they are resolved. (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 boolean is a function that takes two arguments and returns one of them. Define, in pure lambda calculus,

- the Church booleans `TRUE` and `FALSE`
- the function `AND` that takes two Church booleans and returns `TRUE` if they both are `TRUE`, otherwise `FALSE`
- the function `NOT` that takes one Church boolean and returns `TRUE` if it is `FALSE` and `FALSE` if it is `TRUE`.

(7p)

Show some computation steps to obtain the value of `NOT (AND TRUE FALSE)` (3p)