Programming Languages

Exam, 14 January 2008

Course codes: Chalmers TIN321 and TIN 320, GU INN130

Teacher: Aarne Ranta (tel. 1082)

Grading scale: Max = 60p, VG = 5 = 48p, 4 = 36p, G = 3 = 24p. (For TIN 320: 5 = 42p, 4 = 30p, 3 = 20p)

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 at least half of the intermediate are needed to get mark 4. In addition to all the easy and the intermediate, about half of the advanced is needed to get the highest mark (5 or VG).

From another perspective, the easy questions can be answered by anyone who has managed to do the labs without any further reading. The intermediate question may require any material from the lecture notes. The advanced questions may also require material from the book, chapters 1-6.

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: BNFC, 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 less than one page should be enough.

Group 1: easy questions

1. Write a BNF grammar that covers the following kinds of constructs in C/C++/Java:

- variable declarations, consisting of a type followed by a comma-separated non-empty list of variables, each of which can optionally have an initializing expression
- addition and multiplication expressions, both left-associative
- atomic expressions that are identifiers or integer literals
- the types `int` and `double`

It is enough to consider three precedence levels of expressions (from lowest to highest): additions, multiplications, atomic. Parentheses are used, as usual, to lift an expression to the highest level.

You can use the standard BNFC categories `Integer` and `Ident`, and the precedence conventions of BNFC (including `coercions`).

(**10p**)

2. Show the parse tree (**5p**) and the abstract syntax tree (**5p**) of the declaration

```
int x, y = 2, z = x + y * 3
```

in the grammar that you wrote in Question 1. Don't forget to show precedence coercions in the parse tree!

3. Write typing rules or syntax-directed type checking rules for variable declarations as specified in Question 1 (**4p**).

Also write syntax-directed interpretation rules or operational semantic rules for declarations as specified in Question 1. (**4p**).

What error is there in the example of Question 2? How is it detected by the type checker and/or the interpreter? (**2p**).

Group 2: intermediate questions

4. Write a regular expression that recognizes a sequence of tokens that are of one of these two forms, and may be separated by spaces (and doesn't recognize anything else):

- string literal: begins and ends with a double quote `"`, between which it can contain any characters except the double quote itself (so there are no escapes)
- comment: begins with `/*` and ends with `*/`; in between, any characters except the sequence `*/` are allowed; the mark `/*` cannot be inside a string literal

For instance, the automaton should recognize

```
/* comment */ "a string" "/*" /* a comment "*/ "last string"
```

(**5p**),

Write a finite-state automaton that recognizes a list of tokens that are of one of these two forms. You get **3p** if the automaton is non-deterministic (NFA), and **5p**, if it is deterministic (DFA).

Group 3: advanced questions

5. Explain a set of machine instructions that are used in JVM (Java Virtual Machine) to perform additions, multiplications, constants, variables, and assignments, all this for integers only. You need not remember the exact names of these instructions: give just their syntax (**3p**) and small-step operational semantics (**4p**).

Show the code corresponding to the declaration of Question 2 built by using your machine instructions, and assuming that this declaration is the first one in a 0-place function (**3p**).

6. Give operational semantic rules that show the difference between call-by-name and call-by-value lambda calculus (6p).

Show an example of a lambda expression that has different termination behaviours in these two kind of calculi. Give a few evaluation steps to show what this behaviour is in each case (**4p**).