Exam: Models of Computation TDA183, DIT310

Date: Dec 17, 2012, 14.00 - 18.00

Permitted aids: English-Swedish or English-other language dictionary.

Teacher: Bengt Nordström, phone 070 600 1546, Computer Science, University of Gothenburg and Chalmers.

All solutions must be explained! The examination of the course consists of three parts: homework assignments (max 40 points), weekly exercises (max 20 points) and this exam (max 140 points). You have to have 100 points in total in order to pass the course. Solutions to the exam will be available from the homepage of the course.

1. A short (informal) argument that there are functions which cannot be computable is that there are more functions in **N -> N** then programs. This argument can be made more precise:

   (a) (10p) Define what it means for a set to be enumerable.

   (b) (15p) Show that the set of character strings is enumerable.

   (c) (15p) Show that the set **N -> N** is not enumerable.

   ```
   This is covered in the lecture notes "Are all functions
   computable?"and "Hilbert's hotel".
   ```

2. (20p) Beta-conversion in lambda-calculus expresses in some sense that the expression **(\x.e) d** can be rewritten to **e[x:=d]**. Motivate this rule!

   ```
   The question is why this rule look like it does.

   A lambda expression \x.e is standing for the function f defined by
      f x = e
   where e may contain occurrences of x. That the function f is defined by
   the equation above means exactly that
      f d = e[x:=d] for any expression d. This explains the beta-reduction.
   ```

3. (10p) Reduce the following expressions in lambda-calculus to normal form:

   (a) **(\x. (\y.y x)) y**

   (b) **(\x. (\z.z x)) y**

   ```
   This is simple when you have realized that it is necessary to make
   an alpha-conversion in the first problem. The variable y occurrs both
   bound and free. The best alpha-conversion is to convert the first
   problem to the second (yes, they are alpha-convertible!).
   The most common mistake people made in this problem had to do
   ```

```
with parsing: The expression (\z.z x) does not stand for the
identity function applied to x, it stands for
the function which abstracts z from (z x)
```

4. (30p) Suppose that somebody told you that she had written a program **halt :: Bool -> Bool** such that **halt b = True** if the computation of **b** terminates and **halt b = False** otherwise. What would you tell this person? (A correct answer to this problem is a convincing argument that the person is wrong.)

   ```
   This is taken from the lecture notes, it is called the extensional
   halting problem. When correcting this problem, I was amused by the
   number of you who referred to the person above as "he". In fact, a
   majority of references were to a "he" rather than a "she".
   ```

5. (10p) Given an example of a computable function which cannot be expressed in **PRF**! You don't have to give a full proof, you can refer to theorems which we have mentioned in the lectures.

   ```
   In the lecture notes we show that all functions in PRF terminates.
   So, a simple example would be a function which is nowhere defined.
   This is computable (for instance by the Haskell function defined
   by f x = f x). Most people solving this problem had a more complicated
   example (Ackermann's function).
   ```

6. (30p) Give the concrete and abstract syntax of case-expressions in **X** and give the informal and formal semantics. It is not important that you give exactly the same concrete syntax as in the lecture notes.

   ```
   You can find this in the lecture notes about X.
   ```

   Good Luck!

   Bengt