CPSC521: Sample questions for the oral exam

Robin Cockett

29th November 2008

The questions below are a template for the questions I will ask at your oral exam: I may ask slight variations on these questions. You must present the answers to me on the board. It is an open book exam: thus you are permitted to bring, your laptop with any programs you wish to demonstrate. You are also allowed any notes you have prepared for the oral.

I would particularly like to see your last assignment and will be particularly impressed if you can demonstrate the answers to question 1 and 4!

The oral will take approximately one hour and you must arrange a time with me after the 9th December.

20 marks

1. Given the following datatype for terms:

```
data Term = Var Int
          | Node String [Term]
```

   (a) Explain what a *unifier* and what a *most general unifier* of two terms is.

   (b) Explain the basic steps of the unification algorithm:

      i. What is the matching step?

      ii. How does one eliminate a variable?

      iii. What is the "occurs check"?

   (c) Demonstrate the above in finding the most general unifiers of the following (if such exists):

      i. $f(x, g(x, y)) = f(f(z, y), g(w, w))$;

      ii. $f(g(x, y), y) = f(g(y, w), g(z, z))$;

      iii. $f(g(x, y), f(v, w)) = f(z, z)$.

25 marks

2. In the λ-calculus:

   (a) What is a fixed point combinator? Give an example of a fixed point combinator and show that it has the desired property.

   (b) Explain how fixed point combinators are used to encode general recursion.

   (c) Explain how one may represent binary trees

```
data BTree a = Leaf a
             | Node (BTree a) (BTree a)
```

   in the λ-calculus.

   (d) Assuming that one has a representation of numbers and of their basic functions (such as addition) describe how to encode, using the fixed point combinator, the function to sum the leaves of the tree in the λ-calculus.

   Do you need to use general recursion?

20 marks

3. Given an algebraic system with a binary operation g and a unary operation f and rewrites:

$$f(g(x,y)) \rightarrow g(f(x), f(y))$$

$$g(x, g(y,z)) \rightarrow g(g(x,y), z)$$

   (a) What does it mean for a rewrite system to terminate? Does this rewrite system terminate?

   (b) What does it mean for a rewrite system to be locally confluent? What must you check in this system?

   (c) What does it mean for a rewrite system to be confluent? Is this system confluent?

$$\frac{}{x : P, \Gamma \vdash x : P}\ \text{proj}$$

$$\frac{x : P, \Gamma \vdash t : Q}{\Gamma \vdash \lambda x.t : P \to Q}\ \text{abst}$$

$$\frac{\Gamma \vdash f : P \to Q \quad \Gamma \vdash t : P}{\Gamma \vdash (ft) : Q}\ \text{app}$$

Table 1: Type judgments

$$\frac{}{x : P, \Gamma \vdash x : Q[P = Q]}$$

$$\frac{x : P, \Gamma \vdash t : R[\sigma]}{\Gamma \vdash \lambda x.t : Q[\exists P, R.Q = P \to R, \sigma]}$$

$$\frac{\Gamma \vdash f : R[\sigma_1] \quad \Gamma \vdash t : P[\sigma_2]}{\Gamma \vdash (ft) : Q[\exists P, R.R = P \to Q, \sigma_1, \sigma_2]}$$

Table 2: Type judgments with type equations

25 marks

4. Using the judgments for type inference in table 2:

   (a) Show that the term, $\lambda xy.(yx)yx$, cannot be typed in the simply typed lambda calculus.

   (b) Show that $\mathsf{c} = \lambda xyz.xzy$ can be typed in the simply typed lambda calculus and provide its most general type.

10 marks

5. Which of the following are true?

(a) Every λ-calculus term has a normal form: the problem is that with some terms one cannot tell how long it will take to find it.

(b) In the λ-calculus one can decide whether two terms are equal by reducing them to normal form.

(c) The λ-calculus is not quite as expressive as Turing machines which is why these latter are standardly taught in CS-curricula.

(d) If two terms are equal in the λ-calculus they will (eventually) reduce to the same normal form.

(e) The order of evaluation is important as if you do not do a normal order reduction you may not discover the normal form even though there is a one.

(f) By-value reduction is the most efficient evaluation strategy for the λ-calculus.

(g) The confluence of the λ-calculus under β-reduction means that all evaluations of a term will eventually end up at the same place – although some will be more efficient than others..

(h) Functional languages are never used in industry because they are research languages.

(i) You cannot program with state in a functional language unless use the state monad.

(j) Type inference gets in the way of creative programming.