THE UNIVERSITY OF CALGARY

FACULTY OF SCIENCE

FINAL EXAMINATION

COMPUTER SCIENCE 521

December, 2014 Time: 2 hrs.

Instructions

The exam contains questions totaling 100 points. Answer all questions. This exam is closed book.

(15 Marks)

1. Given the following Haskell data type for expressions:

```
data Exp a b = Var a
             | Op b [Exp a b]
```

Write Haskell functions for:

(a) The fold for this data type. Give the type of this function.

(b) The occurs check:

```
occurs:: Eq a => a -> (Exp a b) -> Bool
```

(c) Performing substitution:

```
substitute:: Eq a => [(a,Exp a b)]->(Exp a b)->(Exp a b)
```

(10 marks)

2. Explain what the most general unifier of two terms is.

Which of the following pairs of terms can be unified and what is their most general unifier?

(a) $f(g(x, y), z)$ and $f(w, f(w, x))$

(b) $f(y, h(x, y))$ and $f(g(v, w), h(v, w))$

(c) $f(x, h(v, x))$ and $f(g(z, z), h(w, f(x, v)))$

20 marks

3. In the λ-calculus:

   (a) Give an example of a term with a normal form for which a rightmost innermost rewriting strategy will *not* find the normal form. Explain briefly why a leftmost outermost reduction will find a normal form if there is one.

   (b) Give the de Bruijn form of the term:

   $$\lambda xy.(\lambda xy.(\lambda xy.yx)yx)yx$$

   and give the step-by-step outermost leftmost reduction of the term.

   (c) Explain how the natural numbers can be represented in the λ-calculus - the so called Church numerals. How does one write the predecessor function?

   (d) Explain how one may represent λ-terms in the λ-calculus:

   ```
   data LTerm a = Var a | App LTerm LTerm | Abs Int LTerm
   ```

   Describe the encoding of the constructors, the fold, the map, and the case for this data type.

   (e) In the second recursion theorem one uses a function T such that $T(\underline{X}) = \underline{\underline{X}}$ where $\underline{X}$ is the representation of the λ-term X in the λ-calculus (as above). Describe how T can be implement as a λ-calculus term. (Hint: use the fold above!!)

15 marks

4. Call a λ-term *n-cyclic* if all reduction sequences leaving the term revisit the term (for the first time) after exactly n-steps. Every term is 0-cyclic and, for example, Ω is 1-cyclic.

   (a) Show that the terms

   $$(\lambda yx.xxx)(\lambda yx.xxx)(\lambda yx.xxx)$$

   and

   $$\lambda z.z((\lambda yx.xxy)(\lambda yx.xxy)(\lambda yx.xxy))$$

   are 2-cyclic.

   (b) Show that for each n, there is always a term which is n-cyclic. Furthermore, show that there are always infinitely many terms which are n-cyclic for each n!

   (c) Explain why it is decidable whether a term is n-cyclic but (harder!) undecidable whether a term *reduces* to any particular n-cyclic term. Conclude, for example, that whether a λ-term reduces to Ω cannot be decided.

   Explain your reasoning carefully!

15 marks

5. The basic modern SECD/CES machine has instructions:

   $\mathsf{Clo}(c)$ for pushing a closure of the code c with the current environment on the stack,
   App for perform an application,
   $\#(n)$ for retrieving the n^{th} value in the environment,
   Ret for jumping to the continuation on the stack,
   $\mathsf{Const}(n)$ for pushing the constant n on the stack, and
   Add for addition.

   The machine transitions are:

Before			After		
Code	Env	Stack	Code	Env	Stack
$\mathsf{Clo}(c') : c$	e	s	c	e	$\mathsf{Clos}(c', e) : s$
$\mathsf{App} : c$	e	$\mathsf{Clos}(c', e') : v : s$	c'	$v : e'$	$\mathsf{Clos}(c, e) : s$
$\#(n); c$	e	s	c	e	$e(n) : s$
$\mathsf{Ret} : c$	e	$v : \mathsf{Clos}(c', e') : s$	c'	e'	$v : s$
$\mathsf{Const}(k) : c$	e	s	c	e	$k : s$
$\mathsf{Add} : c$	e	$n : m : s$	c	e	$(n + m) : s$

   Where $\mathsf{Clos}(c, e)$ denotes closure of code c with environment e and $e(n)$ is the n^{th}-element of the environment.

   One way to express the compilation of λ-terms (with arithmetic) into CES-machine code is as follows:

$$\begin{array}{rcl}
[\![\lambda x.t]\!]_s & = & [\mathsf{Clo}([\![t]\!]_{x:s} +\!\!+ [\mathsf{Ret}])] \\
[\![M\ N]\!]_s & = & [\![N]\!]_s +\!\!+ [\![M]\!]_s +\!\!+ [\mathsf{app}] \\
[\![x]\!]_s & = & [\#(n)] \quad \text{where } n = \mathsf{index}\ x\ s \\
[\![k]\!]_s & = & [\mathsf{Const}(k)] \\
[\![a + b]\!]_s & = & [\![b]\!]_s +\!\!+ [\![a]\!]_s +\!\!+ [\mathsf{Add}]
\end{array}$$

   Compile

$$(\lambda xy.x + y)\ 10\ 3$$

   into CES-machine code and show in detail the machine steps for evaluating this code.

   Which reduction strategy does this machine implement? What are the advantages and disadvantages of this reduction strategy?

25 marks

6. Using the judgments for type inference in Table 1 give the result of collecting the type equations and solving the equations (or showing there is no solution) in the following:

   (a) For the term, $\lambda x f.f\ (f\ x)$, in the simply typed lambda calculus (or in **BPCF**), either provide the most general type or show that it cannot be typed.

   (b) Show how the recursive program, fold, fold on lists:

$$\mathsf{fold}\ f\ g\ z = \begin{array}{l} \mathsf{case}\ z \\ \mathsf{of} \left| \begin{array}{lll} \mathsf{nil} & \mapsto & \mathsf{g} \\ \mathsf{cons}\ a\ as & \mapsto & f\ a\ (\mathsf{fold}\ f\ g\ as) \end{array} \right. \end{array}$$

   can be written in **BPCF** as a close term using the fix construct and show how its most general type can be inferred.

$$\frac{}{x : P, z : \Gamma \vdash x : Q \quad \langle P = Q \rangle}\ \text{proj}$$

$$\frac{x : X, z : \Gamma \vdash \quad t : Y \quad \langle E \rangle}{z : \Gamma \vdash \lambda x.t : Q \quad \langle \exists X, Y. Q = X \to Y, E \rangle}\ \text{abst}$$

$$\frac{z : \Gamma \vdash f : Z \quad z : \Gamma \vdash t : X \quad \langle E \rangle}{z : \Gamma \vdash (f t) : Q \quad \langle \exists X, Z. Z = X \to Q, E \rangle}\ \text{app}$$

$$\frac{z : \Gamma \vdash t : Z \quad \langle E \rangle}{z : \Gamma \vdash \mathsf{fix}[t] : Q \quad \langle \exists Z. Z = Q \to Q, E \rangle}\ \text{fix}$$

$$\frac{z : \Gamma \vdash t : X \quad \langle E_1 \rangle \quad z : \Gamma \vdash s : Y \quad \langle E_2 \rangle}{z : \Gamma \vdash (t, s) : Q \quad \langle \exists X, Y. Q = X \times Y, E_1, E_2 \rangle}\ \text{pair}$$

$$\frac{z : \Gamma \vdash t : Z \quad \langle E_1 \rangle \quad z : \Gamma, x : X, y : Y \vdash s : Q \quad \langle E_2 \rangle}{z : \Gamma \vdash \begin{array}{l} \mathsf{case}\ t \\ \mathsf{of}\ (x, y) \mapsto s \end{array} : Q \quad \langle \exists X, Y, Z. Z = X \times Y, E_1, E_2 \rangle}\ \text{pcase}$$

$$\frac{}{z : \Gamma \vdash () : Q \quad \langle Q = 1 \rangle}\ \text{unit}$$

$$\frac{z : \Gamma \vdash t : Z \quad \langle E_1 \rangle \quad z : \Gamma \vdash s : Q \quad \langle E_2 \rangle}{z : \Gamma \vdash \begin{array}{l} \mathsf{case}\ t \\ \mathsf{of}\ () \mapsto s \end{array} : Q \quad \langle \exists Z. Z = 1, E_1, E_2 \rangle}\ \text{ucase}$$

$$\frac{}{z : \Gamma \vdash \mathsf{nil} \ : Q \quad \langle Q = \mathbb{L}(A) \rangle}\ \text{nil}$$

$$\frac{}{z : \Gamma \vdash \mathsf{cons} \ : Q \quad \langle Q = A \times \mathbb{L}(A) \to \mathbb{L}(A) \rangle}\ \text{cons}$$

$$\frac{z : \Gamma \vdash t : X_1 \ \ \langle E_1 \rangle \quad z : \Gamma \vdash t_0 : Y_1 \ \ \langle E_2 \rangle \quad z : \Gamma, v : X_2 \vdash t_1 : Y_2 \ \ \langle E_3 \rangle}{z : \Gamma \vdash \begin{array}{l} \mathsf{case}\ t \\ \mathsf{of} \left| \begin{array}{lll} \mathsf{nil} & \to & t_0 \\ \mathsf{cons}\ v & \to & t_1 \end{array} \right. \end{array} : Q \quad \left\langle \exists \begin{array}{l} X, X_1, \\ Y_1, X_2, \\ Y_2 \end{array} . \begin{array}{l} X_1 = \mathbb{L}(X), \\ X_2 = X \times \mathbb{L}(X), \\ Y_2 = Q, Y_1 = Q, \\ E_1, E_2, E_3 \end{array} \right\rangle}\ \mathbb{L}\ \text{case}$$

Table 1: Rules for type inference

* JRBC