THE UNIVERSITY OF CALGARY

FACULTY OF SCIENCE

FINAL EXAMINATION

COMPUTER SCIENCE 521

December, 2016 Time: 2 hrs.

Instructions

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

(10 Marks)

1. (a) Given the definition of fold(right) in Haskell for lists.

   (b) Use the fold(right) on lists to implement the function

   $$\texttt{inlist} :: \texttt{Eq}\ a \Rightarrow a \to [a] \to \texttt{Bool}$$

   which tests whether an element is in a list.

   (c) What is the foldleft combinator for a list? How do you implement it using a the fold(right) combinator?

(15 marks)

2. (a) Explain what a fixed point combinator is in the λ-calculus.

   (b) Show that $X = (\lambda xy.x\ y\ x)(\lambda yx.y\ (x\ y\ x))$ is a fixed point combinator (this is Tromp's fixed point combinator). Remember to try β-reducing at both ends of the desired equality!

   (c) Consider the general recursive function

   $$\mathsf{nats}\ n = \langle n, \mathsf{nats}\ (n+1)\rangle$$

   where $\langle x, y\rangle := \lambda p.p\ x\ y$. Describe how nats is implemented in the λ-calculus (you may assume a general fixed point combinator Y).

   (d) When is a λ-term in head normal form? Illustrate a head reduction on $\mathsf{nats}\ 0$ as implemented in the λ-calculus in part (c) above.

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:

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

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

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

   (d) One may represent λ-terms in the λ-calculus using the following datatype:

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

   Describe the encoding of the constructors, the fold, and the map 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 using `LTerm Nat`). Describe how T can be implement as a λ-calculus term. (Hint: use the folds 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 $n > 0$, there are always terms which are n-cyclic and are not n-cyclic. Furthermore, show that for each n there are always infinitely many terms which are n-cyclic and infinitely many which are not n-cyclic!

   (c) Explain why it is decidable, for $n > 0$, whether a term is not n-cyclic but (harder!) undecidable whether a term *never reduces* to any n-cyclic term.

   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

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

   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, $\lambda x f.(f\ x)\ (x\ f)$, 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, map, map on lists:

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

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

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

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

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

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

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

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

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

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

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

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

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

Table 1: Rules for type inference

* JRBC