Programming Language Qualifier

January 1997

1. Answer the following questions about inheritance.

   (a) Your friend Java Nagila is learning English, and she is puzzled about the following sentence: "*The platypus is an egg-laying mammal.*" Fortunately, she is an expert in object-oriented languages. Explain the sentence to her, using object-oriented concepts such as superclass, subclass, and method overriding. Your answer should be less than 30 words long.

   (b) Consider expressions like (2+3)*(5-4) in which all operators are *binary* operators — that is, they have two operands which are themselves either expressions or integer constants. In an OO language, expression trees can be represented by objects which are instances of the following class:

   ```
   class ExpTree {
     Operator Op; // assume there is a class Operator
     Integer Konstant;
     ExpTree Left;
     ExpTree Right;
     ...
   }
   ```

   This class has the disadvantage that `ExpTree` objects have storage allocated in them for both operators and constants, even though a given `ExpTree` object has valid data in either the `Op` field (internal nodes of the tree) or the `Konstant` field (leaves of the tree), but not both.

   Design a class (or classes) for representing binary expression trees without this waste of storage (that is, so that the internal nodes do not have storage allocated for a constant, and leaf nodes do not have storage allocated for an operator.)

2. A symmetric matrix $A[0..n-1, 0..n-1]$ is a square matrix in which $A[i,j] = A[j,i]$ for all i and j between 0 and $n-1$. A symmetric matrix can be represented compactly by storing only the lower triangular portion of the matrix as shown below. Assume that the matrix is stored in row-major order, that each element fits into one word of memory, and that $A[0,0]$ is stored at memory location 0. Write down an expression for the address where element $A[k,m]$ is stored. Note that k may be either larger or smaller than m.

$$\begin{pmatrix} \boxed{3} & 4 & 7 & 9 & 11 \\ \boxed{4} & \boxed{5} & 0 & 5 & 0 \\ \boxed{7} & \boxed{0} & \boxed{8} & 4 & 34 \\ \boxed{9} & \boxed{5} & \boxed{4} & \boxed{9} & 6 \\ \boxed{11} & \boxed{0} & \boxed{34} & \boxed{6} & \boxed{12} \end{pmatrix}$$

3. A *possibly uninitialized* variable is a variable x such that in the control-flow graph of the program, there is a path P from START to a use of x such that no definition of x occurs on the path. Explain how the set of possibly uninitialized variables in a program can be determined by dataflow analysis. Your answer must give a brief description of:

   (a) the structure you are assuming for control-flow graphs,

   (b) the partially ordered set in your dataflow analysis,

   (c) the schema for your dataflow equations.

4. The following BNF grammar defines the abstract syntax of a *call-by-name* programming language called λ-bool. Let x range over a set of variables, and let 0 and 1 be Boolean values (say 1 is true):

$$\begin{array}{llll} \text{(types)} & \tau & ::= & \mathsf{bool} \mid \tau_1 \to \tau_2 \\ \text{(terms)} & M & ::= & x \mid 0 \mid 1 \mid \lambda x{:}\tau.M \mid M_1\, M_2 \mid \\ & & & \texttt{if } M_1 \texttt{ then } M_2 \texttt{ else } M_3 \\ \text{(values)} & V & ::= & 0 \mid 1 \mid \lambda x{:}\tau.M \end{array}$$

The evaluation rules are expressed in terms of the relation $M \Downarrow V$ where V is a subset of the terms called values. The relation is defined using the following inference rules, where $[V/x]M$ is the term obtained by performing capture-avoiding substitution of the value V for the free occurrences of x within M:

$$V \Downarrow V \qquad \frac{M_1 \Downarrow \lambda x{:}\tau.M \qquad [M_2/x]M \Downarrow V}{M_1\, M_2 \Downarrow V}$$

$$\frac{M_1 \Downarrow 0 \qquad M_3 \Downarrow V}{\texttt{if } M_1 \texttt{ then } M_2 \texttt{ else } M_3 \Downarrow V} \qquad \frac{M_1 \Downarrow 1 \qquad M_2 \Downarrow V}{\texttt{if } M_1 \texttt{ then } M_2 \texttt{ else } M_3 \Downarrow V}$$

The typing rules are given by typing judgments of the form $\Gamma \vdash M : \tau$, where Γ is a partial function from variables to types. The judgments are derived by the following inference rules:

$$\textbf{(var)}\ \Gamma \vdash x : \Gamma(x) \qquad \textbf{(ff)}\ \Gamma \vdash 0 : \mathsf{bool} \qquad \textbf{(tt)}\ \Gamma \vdash 1 : \mathsf{bool}$$

$$\textbf{(abs)}\ \frac{\Gamma[x{:}\tau_1] \vdash M : \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.M : \tau_1 \to \tau_2} \qquad \textbf{(app)}\ \frac{\Gamma \vdash M_1 : \tau' \to \tau \quad \Gamma \vdash M_2 : \tau'}{\Gamma \vdash M_1\, M_2 : \tau}$$

$$\textbf{(if)}\ \frac{\Gamma \vdash M_1 : \mathsf{bool} \quad \Gamma \vdash M_2 : \tau \quad \Gamma \vdash M_3 : \tau}{\Gamma \vdash \texttt{if } M_1 \texttt{ then } M_2 \texttt{ else } M_3 : \tau}$$

Let Fun(bool, bool) be the set of *total* functions from Booleans to Booleans. Suppose we represent partial functions from Booleans to Booleans as total functions from $\{0, 1, \bot\}$ to $\{0, 1, \bot\}$ where $f(x) = \bot$ means f is undefined at x and $f(\bot) = y$ means that f provides output without evaluating its input. Let PFun(bool, bool) denote these partial functions from (lifted) Booleans to (lifted) Booleans.

(a) How many elements are there in $\mathrm{Fun}(\mathsf{bool}, \mathsf{bool})$ and in $\mathrm{PFun}(\mathsf{bool}, \mathsf{bool})$?

(b) Do the above rules determine whether $\mathsf{bool} \to \mathsf{bool}$ terms are elements of $\mathrm{Fun}(\mathsf{bool}, \mathsf{bool})$ or of $\mathrm{PFun}(\mathsf{bool}, \mathsf{bool})$? That is, should we choose $\mathrm{Fun}(\mathsf{bool}, \mathsf{bool})$ or $\mathrm{PFun}(\mathsf{bool}, \mathsf{bool})$ as the semantic interpretation of $\mathsf{bool} \to \mathsf{bool}$?

Now consider the following extension of λ-bool called λ-`fix` where we add recursive terms of the form $\mathtt{fix}(x{:}\tau.M)$:

$$\frac{[(\mathtt{fix}(x{:}\tau.M))/x]M \Downarrow V}{\mathtt{fix}(x{:}\tau.M) \Downarrow V}$$

$$(\text{fix}) \quad \frac{\Gamma[x{:}\tau] \vdash M : \tau}{\Gamma \vdash \mathtt{fix}(x{:}\tau.M) : \tau}$$

(c) Are there expressions in the extended language for all the partial functions in $\mathrm{Pfun}(\mathsf{bool}, \mathsf{bool})$? That is, for each partial function f in $\mathrm{Pfun}(\mathsf{bool}, \mathsf{bool})$, is there a well-formed term M whose semantic interpretation should be f?

(d) Do the rules determine whether $\mathsf{bool} \to \mathsf{bool}$ is $\mathrm{Fun}(\mathsf{bool}, \mathsf{bool})$ or not? That is, can we choose $\mathrm{Fun}(\mathsf{bool}, \mathsf{bool})$ as the interpretation of $\mathsf{bool} \to \mathsf{bool}$?

5. Consider the extension of the original λ-bool language called λ-Top where we add a new type called Top and three new terms, $\mathtt{bool2top}(M)$, $\mathtt{arrow2top}(M)$, and `case`:

$$\begin{array}{llll}
\text{(types)} & \tau & ::= & \mathsf{bool} \mid \tau_1 \to \tau_2 \mid \mathsf{Top} \\
\text{(terms)} & M & ::= & \cdots \mid \mathtt{bool2top}(M) \mid \mathtt{arrow2top}(M) \mid \\
 & & & \mathtt{case}\ M\ \mathtt{of}\ \mathtt{bool}(x_1)\ \mathtt{=>}\ M_1\ \mathtt{or}\ \mathtt{arrow}(x_2)\ \mathtt{=>}\ M_2 \\
\text{(values)} & V & ::= & 0 \mid 1 \mid \lambda x{:}\tau.M \mid \mathtt{bool2top}(M) \mid \mathtt{arrow2top}(M)
\end{array}$$

Intuitively, Top corresponds to the recursive sum type $\mu\alpha.(\mathsf{bool} + (\alpha \to \alpha))$. Another way to understand Top is to consider the corresponding ML datatype definition:

```
datatype Top = bool2top of bool | arrow2top of Top -> Top
```

Informally, `bool2top` provides a way to "coerce" a Boolean to a Top value, and `arrow2top` provides a way to "coerce" a function of type Top → Top to a Top value. The `case` construct provides a way to examine a Top value to see whether it is a coerced Boolean or function, and then extract this underlying value.

The evaluation and typing rules for the new constructs are given below:

$$\frac{M \Downarrow V}{\mathtt{bool2top}(M) \Downarrow \mathtt{bool2top}(V)} \qquad \frac{M \Downarrow V}{\mathtt{arrow2top}(M) \Downarrow \mathtt{arrow2top}(V)}$$

$$\frac{M \Downarrow \mathtt{bool2top}(V') \quad [V'/x_1]M_1 \Downarrow V}{\mathtt{case}\ M\ \mathtt{of}\ \mathtt{bool}(x_1)\ \mathtt{=>}\ M_1\ \mathtt{or}\ \mathtt{arrow}(x_2)\ \mathtt{=>}\ M_2 \Downarrow V}$$

$$\frac{M \Downarrow \mathtt{arrow2top}(V') \quad [V'/x_2]M_2 \Downarrow V}{\mathtt{case}\ M\ \mathtt{of}\ \mathtt{bool}(x_1)\ \mathtt{=>}\ M_1\ \mathtt{or}\ \mathtt{arrow}(x_2)\ \mathtt{=>}\ M_2 \Downarrow V}$$

$$(\text{b2t}) \quad \frac{\Gamma \vdash M : \mathsf{bool}}{\Gamma \vdash \mathtt{bool2top}(M) : \mathsf{Top}} \qquad (\text{a2t}) \quad \frac{\Gamma \vdash M : \mathsf{Top} \to \mathsf{Top}}{\Gamma \vdash \mathtt{arrow2top}(M) : \mathsf{Top}}$$

$$(\text{case})\ \frac{\Gamma \vdash M : \mathsf{Top} \quad \Gamma[x_1{:}\mathsf{bool}] \vdash M_1 : \tau \quad \Gamma[x_2{:}\mathsf{Top} \to \mathsf{Top}] \vdash M_2 : \tau}{\Gamma \vdash \texttt{case}\ M\ \texttt{of bool(}x_1\texttt{) => }M_1\ \texttt{or arrow(}x_2\texttt{) => }M_2 : \tau}$$

(a) Give a typing derivation that shows that $\Gamma \vdash M_0 : \mathsf{Top} \to \mathsf{Top}$, where Γ is the empty type assignment and the term M_0 is defined below:

$$M_0 \overset{\text{def}}{=} \lambda x{:}\mathsf{Top}.\texttt{case}\ x\ \texttt{of bool(}x_1\texttt{) => }0\ \texttt{or arrow(}x_2\texttt{) => }(x_2\, x)$$

(b) Is the following term (where M_0 is taken from part (a)) well-typed and if so, what is its behavior when evaluated?

$$M_0\,(\texttt{arrow2top}(M_0))$$

6. The abstract syntax of the *untyped* lambda calculus with Booleans is given by the following grammar:

$$(\text{terms})\quad U \ ::=\ x \mid 0 \mid 1 \mid \lambda x.U \mid U_1\, U_2 \mid \texttt{if}\ U_1\ \texttt{then}\ U_2\ \texttt{else}\ U_3$$

The evaluation rules for this language are the same as those given for λ-bool, except that we replace all Ms with Us and erase the type labelling the bound variable of λ-terms. For instance:

$$\frac{U_1 \Downarrow \lambda x.M \qquad [U_2/x]U \Downarrow U}{U_1\, U_2 \Downarrow V}$$

The function $\mathcal{C}[\![\text{-}]\!]$ defined below maps untyped lambda terms to λ-Top terms:

$\mathcal{C}[\![x]\!] = x$
$\mathcal{C}[\![0]\!] = \texttt{bool2top}(0)$
$\mathcal{C}[\![1]\!] = \texttt{bool2top}(1)$
$\mathcal{C}[\![\texttt{if}\ U_1\ \texttt{then}\ U_2\ \texttt{else}\ U_3]\!] =$
$\quad \texttt{if (case}\ \mathcal{C}[\![U_1]\!]\ \texttt{of bool(}x_1\texttt{) => }x_1\ \texttt{or arrow(}x_2\texttt{) => }0\texttt{)}\ \texttt{then}\ \mathcal{C}[\![U_2]\!]\ \texttt{else}\ \mathcal{C}[\![U_3]\!]$
$\mathcal{C}[\![\lambda x.U]\!] = \lambda x{:}\mathsf{Top}.\mathcal{C}[\![U]\!]$
$\mathcal{C}[\![U_1\, U_2]\!] =$
$\quad (\texttt{case}\ \mathcal{C}[\![U_1]\!]\ \texttt{of bool(}x_1\texttt{) => }(\lambda x_3{:}\mathsf{Top}.x_3)\ \texttt{or arrow(}x_2\texttt{) => }x_2)\,(\mathcal{C}[\![U_2]\!])$

(a) Prove by structural induction on U that if U has free variables $\{x_1, \cdots, x_n\}$ and $\Gamma(x_i) = \mathsf{Top}$ for $1 \le i \le n$, then $\Gamma \vdash \mathcal{C}[\![U]\!] : \mathsf{Top}$.

(b) Let U be a closed term that that gets stuck during evaluation because of a run-time type error. For instance, evaluation of $U_1\, U_2$ might get stuck because U_1 evaluates to a Boolean instead of a λ-abstraction. Does evaluation of $\mathcal{C}[\![U]\!]$ get stuck due to a type error?