CS611 Take Home Final Exam

Greg Morrisett

Due Friday, December 5, 1997 at 3:00pm

This is the final exam for CS611. You are to work *alone*, though you may refer to notes and homeworks from class, and Winskell's book. If you have any questions, please contact Steve (`zdance@cs.cornell.edu`) or me (`jgm@cs.cornell.edu`). *Do not discuss the exam with anyone else.*

Submit your written results to my administrative assistant, Joan Lockwood in Upson 4147 by 3pm. Make sure your name is on every sheet of paper that you turn in. For problem 2(e), submit a printout of your ML program.

1. Short Answer

   (a) Formally explain what we mean by a "type safe language". Is Java a type-safe language? Why or why not?

   (b) Suppose C_1 and C_2 are each commands of IMP. Briefly describe two different techniques for proving or disproving that the two commands are "equal". You should appeal to formalism that we developed in class.

   (c) Suppose $\lambda x{:}\tau_a.e_1$ and $\lambda y{:}\tau_a.e_2$ are each simply-typed lambda calculus functions with type $\tau_a \rightarrow \tau_b$. Briefly describe two different techniques for proving or disproving that the two functions are "equal". You should appeal to formalism that we developed in class.

   (d) Briefly explain the role that CPOs play in programming language semantics. In particular, why did we choose to interpret PCF types as pointed CPOs?

   (e) True or false: Subtyping is a matter of opinion.

   (f) What does it mean for a language to be "confluent" and why is this an important property from a semantic point of view?

2. Semantics of Regular Expressions

   Suppose we are given a BNF definition for the abstract syntax of the language of regular expressions:

   $$R \quad ::= \quad \epsilon \mid c \mid R_1\,R_2 \mid R_1 + R_2 \mid R^*$$

   where ϵ denotes the empty string, c is drawn from some finite alphabet Σ, $R_1\,R_2$ denotes concatentation, $R_1 + R_2$ denotes alternation (*i.e.*, either R_1 or R_2), and R^* denotes the Kleene star (*i.e.*, zero or more repititions of R).

   (a) Define a denotational semantics $\mathcal{R}[\![\;]\!]$ that maps regular expressions to the set of strings which they represent. For example, the regular expression $(\mathtt{ab})^*$ represents the set $\{\epsilon, \mathtt{ab}, \mathtt{abab}, \mathtt{ababab}, \cdots\}$. Make sure to justify your construction by establishing the necessary conditions for any theorems or lemmas you use.

(b) The rewriting system in Figure 1 defines a "large-step" semantics for regular expressions. Configurations are of the form $\langle R, s, K\rangle$ where R is a regular expression, s is a string over the alphabet Σ, and K is a "stack" or list of regular expressions:

$$\begin{array}{lrcl} \text{(strings)} & s & ::= & \epsilon \mid c :: s \\ \text{(stacks)} & K & ::= & \text{nil} \mid R :: K \end{array}$$

The semantics defines a relation of the form $\langle R, s, K\rangle \Downarrow b$ where b is a boolean value (either true or false). The idea is that configuration $\langle R, s, \text{nil}\rangle$ should evaluate to true iff s is in the set $\mathcal{R}[\![R]\!]$. The "stack" K is used to keep track of regular expressions that have yet to be matched.

Using your definition of $\mathcal{R}[\![\,]\!]$, prove that if $\langle R, s, \text{nil}\rangle \Downarrow b$ then b is true iff $s \in \mathcal{R}[\![R]\!]$.

(c) Suppose you were asked to prove by induction on R that for all $s \in \mathcal{R}[\![R]\!]$, $\langle R, s, \text{nil}\rangle \Downarrow$ true. Where would you run into trouble?

(d) Show, by giving a counter example, that it is not the case that $s \in \mathcal{R}[\![R]\!]$ implies $\langle R, s, \text{nil}\rangle \Downarrow$ true.

(e) Implement the given operational semantics as an ML function with the signature:

```
val match : reg -> (char list) -> (char list -> bool) -> bool
```

where the datatype `reg` is defined as:

```
datatype reg = Epsilon | Char of char | Concat of reg * reg |
   Plus of reg * reg | Star of reg
```

Note that the "stack" is represented as a function taking a character list and returning a bool instead of as a list of regular expressions. Your function should have the property that if $\langle R, s, \text{nil}\rangle \Downarrow b$ then the ML expression `match` R s `(fn [] => true | _ => false)` should evaluate to the boolean b.

3. Continuations

Figure 2 gives the abstract syntax, static semantics, and operational semantics for the call-by-value simply-typed lambda calculus extended with continuations. Assume the following lemma is true (it is):

Lemma 0.1 (Substitution) *If $\Gamma, x{:}\tau' \vdash e : \tau$ and $\Gamma \vdash e' : \tau'$ then $\Gamma \vdash e\{e'/x\} : \tau$.*

Prove the following Preservation and Progress lemmas to establish type soundness for the language:

(a) If $\vdash e : \text{int}$, $e = E[I]$ and $E[I] \longmapsto e'$, then $\vdash e' : \text{int}$.

(b) If $\vdash e : \text{int}$ then either $e = i$ for some integer i or else there exists an E, I, and e' such that $e = E[I]$ and $E[I] \longmapsto e'$.

(c) Suppose we did not restrict programs to have integer type. Give an example expression such that $\vdash e : \tau$, $e \longmapsto e'$, but $\not\vdash e' : \tau$.

$$\langle \epsilon, \epsilon, \mathrm{nil} \rangle \Downarrow \mathrm{true}$$

$$\langle \epsilon, c :: s, \mathrm{nil} \rangle \Downarrow \mathrm{false}$$

$$\frac{\langle R, s, K \rangle \Downarrow b}{\langle \epsilon, s, R :: K \rangle \Downarrow b}$$

$$\langle c, \epsilon, K \rangle \Downarrow \mathrm{false}$$

$$\langle c, c' :: s, K \rangle \Downarrow \mathrm{false} \ (c \neq c')$$

$$\frac{\langle \epsilon, s, K \rangle \Downarrow b}{\langle c, c :: s, K \rangle \Downarrow b}$$

$$\frac{\langle R_1, s, R_2 :: K \rangle \Downarrow b}{\langle R_1 R_2, s, K \rangle \Downarrow b}$$

$$\frac{\langle R_1, s, K \rangle \Downarrow b_1 \qquad \langle R_2, s, K \rangle \Downarrow b_2}{\langle R_1 + R_2, s, K \rangle \Downarrow b} (b = b_1 \vee b_2)$$

$$\frac{\langle \epsilon + (RR^*), s, K \rangle \Downarrow b}{\langle R^*, s, K \rangle \Downarrow b}$$

Figure 1: Large-Step Semantics for Regular Expressions

$$
\begin{array}{llll}
\text{(types)} & \tau & ::= & \mathrm{int} \mid \tau_1 \rightarrow \tau_2 \mid \mathrm{cont}(\tau) \\
\text{(expressions)} & e & ::= & v \mid x \mid e_1\, e_2 \mid \texttt{callcc}(e) \mid \texttt{throw}(e_1, e_2) \\
\text{(values)} & v & ::= & i \mid \lambda x{:}\tau.e \mid \texttt{cont}(\lambda x{:}\tau.E[x]) \\
\text{(contexts)} & E & ::= & [\,] \mid E\, e \mid v\, E \mid \texttt{callcc}(E) \mid \texttt{throw}(E, e) \mid \texttt{throw}(v, E) \\
\text{(instructions)} & I & ::= & (\lambda x{:}\tau.e)\, v \mid \texttt{callcc}(\lambda x{:}\mathrm{cont}(\tau).e) \mid \texttt{throw}(\texttt{cont}(\lambda x{:}\tau.E[x]), v)
\end{array}
$$

$$
(\textbf{int})\ \Gamma \vdash i : \mathrm{int} \qquad\qquad (\textbf{var})\ \Gamma \vdash x : \Gamma(x)
$$

$$
(\textbf{fn})\ \frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.e : \tau_1 \rightarrow \tau_2} \qquad\qquad (\textbf{app})\ \frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\, e_2 : \tau_2}
$$

$$
(\textbf{cont})\ \frac{\Gamma, x{:}\tau \vdash E[x] : \mathrm{int}}{\Gamma \vdash \texttt{cont}(\lambda x{:}\tau.E[x]) : \mathrm{cont}(\tau)} \quad (x \notin FV(E))
$$

$$
(\textbf{callcc})\ \frac{\Gamma \vdash e : \mathrm{cont}(\tau) \rightarrow \tau}{\Gamma \vdash \texttt{callcc}(e) : \tau}
$$

$$
(\textbf{throw})\ \frac{\Gamma \vdash e_1 : \mathrm{cont}(\tau') \qquad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash \texttt{throw}(e_1, e_2) : \tau}
$$

$$
E[(\lambda x{:}\tau.e)\, v] \longmapsto E[e\{v/x\}]
$$

$$
E[\texttt{callcc}(\lambda x{:}\mathrm{cont}(\tau).e)] \longmapsto E[(\lambda x{:}\mathrm{cont}(\tau).e)\, (\texttt{cont}(\lambda x{:}\tau.E[x]))]
$$

$$
E[\texttt{throw}(\texttt{cont}(\lambda x{:}\tau.E'[x]), v)] \longmapsto E'[v]
$$

Figure 2: Syntax, Static & Dynamic Semantics of STLC+Cont