May 16, 1991

CS 541 — Programming Languages 1

Final Exam

Please write your name here:

Before you begin, please take a moment to look over the entire test so that you can budget your time.

Answers to essay questions will be judged on their completeness and clarity. However, please do not feel obligated to fill up all the empty space on the page (especially for questions that are not worth many points). Conversely, do not feel limited by the space on a page, since you may continue an answer on the back of a page if you wish.

problem	subject	points	score
1	OO Languages vs. Functional langauges	20	
2	Recursive definitions	20	
3	Compute the fixed point	20	
4	Semantics of call/cc	20	
5	Semantics of Smalltalk	40	
6	Expressive power	15	
7	Semantics of Modules	15	
8	Denotational and Operational Semantics	50	
Total		200	

1. (20 points) Describe the essential differences between the denotational semantics of an object-oriented and a functional language. Be specific.

2. (20 points) Explain (in words) how one gives meaning to a recursive function definition in denotational semantics and the conditions under which such a meaning is defined.

3. Consider the following recursive function definition. $g : Nat_\perp \to Nat_\perp$ recursively defined by

$$g = \underline{\lambda} n \,.\, plus\ n\ ((equals\ n\ zero) \to zero [\!] g\ (g\ (minus\ n\ one)))$$

(a) (14 points) Build the generator (i.e., functional) G associated with g for purposes of computing its fixedpoint and show the graphs (set representations) of the functions $G^1(\perp)$, $G^2(\perp)$, $G^3(\perp)$, $G^4(\perp)$. (For example, the graph of *plus* 3 is $\{(n, n+3) \mid n \in Nat\}$.)

(b) (6 points) What is $(g\ 3)$?

4. (20 points) Explain how the `call-with-current-continuation` (henceforth, call/cc) primitive of Scheme allows Scheme to simulate arbitrary control structures. Relate your explanation to the denotational semantics of control structures.

5. Some questions on the semantics of Smalltalk.

   (a) (5 points) What is the parameter passing mechanism in Smalltalk?

   (b) (5 points) Describe what a block is in Smalltalk. What other construct in Scheme most closely resembles a Smalltalk block?

   (c) (15 points) What does message passing mean in an object-oriented langauge? That is, give an overview of the denotational semantics of message passing.

   (d) (5 points) What scope rule applies to message names in Smalltalk?

6. (15 points) Why is it difficult to compare the expressive power of different programming languages? What ideas may help overcome these differences?

7. (15 points) Briefly explain the semantics of non-hierarchical scopes (modules). How does the provision of such non-hierarchical scopes aid information hiding?

8. Consider the following small language, whose grammar is given below. We omit the grammar for identifiers $\langle id \rangle$.

$$
\begin{array}{lll}
\langle expr \rangle & ::= & \texttt{true} \mid \texttt{false} \mid \langle id \rangle \\
 & \mid & \texttt{if } \langle expr \rangle \texttt{ then } \langle expr \rangle \texttt{ else } \langle expr \rangle \texttt{ fi} \\
 & \mid & \texttt{(nu } \langle id \rangle \texttt{ . } \langle expr \rangle \texttt{)}
\end{array}
$$

The operational semantics of this language is be given by the following TTS. The set of configurations, Γ is the set of expressions $\langle expr \rangle$. The set of final configurations $T = \{\texttt{true}, \texttt{false}\}$. The input and output functions are the identity functions.

The following axioms and inference rules define $\Rightarrow$.

- $\texttt{if true then } e_2 \texttt{ else } e_3 \texttt{ fi} \Rightarrow e_2$
- $\texttt{if false then } e_2 \texttt{ else } e_3 \texttt{ fi} \Rightarrow e_3$
- $\texttt{(nu x . } e_1\texttt{)} \Rightarrow [\texttt{(nu x . } e_1\texttt{)/x}]e_1$
- An inference rule for reducing the test of an if.

$$
\frac{e \Rightarrow e'}{\texttt{if } e \texttt{ then } e_2 \texttt{ else } e_3 \texttt{ fi} \Rightarrow \texttt{if } e' \texttt{ then } e_2 \texttt{ else } e_3 \texttt{ fi}}
$$

In a term of the form $\texttt{(nu x . } e\texttt{)}$, $\texttt{x}$ is considered to be a bound variable. Thus the syntactic substitution $[e_2/\texttt{x}]e_1$ used above is defined as follows:

$$
\begin{array}{lll}
[e_2/\texttt{x}]\texttt{true} & = & \texttt{true} \\
[e_2/\texttt{x}]\texttt{false} & = & \texttt{false} \\
[e_2/\texttt{y}]\texttt{x} & = & \begin{cases} e_2 & \text{if } \texttt{x} \equiv \texttt{y} \\ \texttt{x} & \text{otherwise} \end{cases} \\
[e_4/\texttt{x}]\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \texttt{ fi} & = & \texttt{if } [e_4/\texttt{x}]e_1 \texttt{ then } [e_4/\texttt{x}]e_2 \texttt{ else } [e_4/\texttt{x}]e_3 \texttt{ fi} \\
[e_2/\texttt{y}]\texttt{(nu x . } e_1\texttt{)} & = & \begin{cases} \texttt{(nu x . } e_1\texttt{)} & \text{if } \texttt{x} \equiv \texttt{y} \\ \texttt{(nu x . } [e_2/\texttt{y}]e_1\texttt{)} & \text{otherwise} \end{cases}
\end{array}
$$

(a) (5 points) Is the meaning of the term $\texttt{y}$ defined by the above semantics? If so, what is it?

(b) (5 points) Is the meaning of the term $\texttt{(nu x . x)}$ defined by the above semantics? If so, what is it?

As a start on the denotational semantics of this language, the following is its abstract syntax.

Abstract Syntax:
P $\in$ Program
E $\in$ Expression
I $\in$ Identifier

P ::= E
E ::= `true` | `false` | I
  | `if` E_1 `then` E_2 `else` E_3 `fi`
  | `(nu` I `.` E`)`

(c) (15 points) Now you fill in the missing parts of the domain equations for a direct syntax. (You may fill in anywhere there is a blank.) You should read over the next part of this question before you commit yourself to particular domains, so that everthing will type check. Also fill in any auxiliary operations on the domains you may need below while working on the next part.

Domains $b \in \mathit{Boolean}$
$v \in \mathit{Value} =$

$e \in \mathit{Expressible\text{-}value} =$
$\mathit{ANSWER} =$
$\mathit{Error} = \mathit{Unit}$

$d \in \mathit{Denotable\text{-}value} =$
$u \in \mathit{Environment} = \text{Identifier} \rightarrow (\mathit{Denotable\text{-}value} + \mathit{Unbound})$
$\mathit{Unbound} = \mathit{Unit}$

Operations
emptyenv, *accessenv* and *updateenv* as in Schmidt

(d) (25 points) Complete the definition of the valuation function $\mathcal{E}$ by filling in the missing clauses.

Valuation functions:
$\mathcal{P}$: Program $\rightarrow$ *ANSWER*
$\mathcal{P}[\![\mathrm{E}]\!] = \mathcal{E}[\![\mathrm{E}]\!]\ \mathit{emptyenv}$

$\mathcal{E}$: Expression $\rightarrow$ *Environment* $\rightarrow$ *ANSWER*
$\mathcal{E}[\![\texttt{true}]\!] = \lambda u\ .\ \mathrm{in}\mathit{Expressible\text{-}value}(\mathit{true})$
$\mathcal{E}[\![\texttt{false}]\!] = \lambda u\ .\ \mathrm{in}\mathit{Expressible\text{-}value}(\mathit{false})$
$\mathcal{E}[\![\mathrm{I}]\!] = \lambda u\ .$
$\quad$ **cases** $(\mathit{accessenv}[\![\mathrm{I}]\!]\ u)$ **of**
$\quad\quad \mathrm{is}\mathit{Denotable\text{-}value}(d) \ \rightarrow\ \mathrm{in}\mathit{Expressible\text{-}value}(d)$
$\quad\quad \mathrm{is}\mathit{Unbound}() \ \rightarrow\ \mathrm{in}\mathit{Error}()$
$\quad$ **end**
$\mathcal{E}[\![\texttt{if}\ \mathrm{E}_1\ \texttt{then}\ \mathrm{E}_2\ \texttt{else}\ \mathrm{E}_3\ \texttt{fi}]\!] =$

$\mathcal{E}[\![\texttt{(nu}\ \mathrm{I}\ \texttt{.}\ \mathrm{E)}]\!] =$