Homework 6

This homework is due Monday, April 9 (in class).

1. Consider the lifted flat domain $D = \{a, b, c\}_\perp$, where $\sqsubseteq_D$ relates $\perp$ to a, to b, and to c, but the proper elements are not related to each other. State the reasons for your answers.
   (a) (2 points) How many monotonic functions are there from D to D?
   (b) (2 points) How many continuous functions are there from D to D?

2. Design a functional language based on the lambda calculus that can perform computations over booleans and natural numbers along the following lines.
   - Programs consist of a (possibly empty) sequence of nonrecursive function definitions, followed by an expression which may use those functions as well as the operations of the built-in types and other language-defined expressions. There is no "input", other than the expression itself.
   - Instead of λ, function abstraction will be indicated by the keyword `fun`.
   - The language should pass arguments by value.
   - Function abstractions are *not* first-class objects, that is they may only be used in expressions by calling them and may not be passed as arguments.
   - There should be an `if` expression.

   Assume that a set of binary operations the natural numbers, such as `=` and `+`, have already been agreed on. These operations are defined on all arguments of the appropriate domains. You do not have to design these operations.

   The following is the abstract syntax. (The literals include both boolean and numeric literals; you do not have to give their syntax.)

   $P \in$ Program
   $E \in$ Expression
   $F \in$ Function-abstract
   $I \in$ Identifier
   $L \in$ Literal
   $O \in$ Operator-for-Built-in

   P ::= `fun` I_1 `(`I_2`)` `=` E `;` P | E
   E ::= L
   | I
   | `(`O E_1 E_2`)`
   | `(if` E_1 E_2 E_3`)`
   | `(`I E_2`)`
   | `(`F E_2`)`
   F ::= `(fun` I E_1`)`

(a) The following domain equations will be used for giving a standard semantics to this language.

$$
\begin{array}{rcl}
n & \in & \mathit{Nat} \\
b & \in & \mathit{Boolean} \\
p & \in & \mathit{Procedure} = \mathit{Denotable\text{-}value} \to \mathit{Expcont} \to \mathit{ANSWER} \\
v & \in & \mathit{Value} = \mathit{Nat} + \mathit{Boolean} \\
e & \in & \mathit{Expressible\text{-}value} = \mathit{Value} \\
d & \in & \mathit{Denotable\text{-}value} = \mathit{Value} + \mathit{Procedure} \\
 & \in & \mathit{ANSWER} = (\mathit{Expressible\text{-}value} + \mathit{Error})_{\perp} \\
 & \in & \mathit{Error} = \mathit{Unit} \\
u & \in & \mathit{Environment} = \text{Identifier} \to (\mathit{Denotable\text{-}value} + \mathit{Unbound}) \\
 & \in & \mathit{Unbound} = \mathit{Unit} \\
k & \in & \mathit{Expcont} = \mathit{Expressible\text{-}value} \to \mathit{ANSWER}
\end{array}
$$

Note that the domains *Expressible-value* and *Denotable-value* are changed from the exam.

i. (3 points) Why is the domain *Expressible-value* different from the domain *Denotable-value*? (That is, what is the reason for the particular choices given above?)

ii. (5 points) Why are they different than on the exam? (That is, why was the exam wrong?)

iii. (3 points) The above domain equations are for a standard semantics. What would the domain equation for *Procedure* be if one wished to give a direct semantics (i.e., not using continuations) instead of a standard semantics for this language?

(b) (4 points) Consider the following auxiliary functions.

$\mathit{extend\text{-}env} : \mathit{Environment} \to \text{Identifier} \to \mathit{Denotable\text{-}value} \to \mathit{Environment}$

$\quad \mathit{extend\text{-}env} = \lambda u \,.\, \lambda I \,.\, \lambda d \,.$

$\qquad \lambda I' \,.\, (\mathit{same\text{-}identifier?}\, I\ I') \to (\text{in}\mathit{Denotable\text{-}value}\ d) \,[\!]\, (u\ I')$

$\mathit{lookup} : \mathit{Environment} \to \text{Identifier} \to (\mathit{Denotable\text{-}value} + \mathit{Unbound})$

$\quad \mathit{lookup} = \lambda u \,.\, \lambda I \,.\, (u\ I)$

$\mathit{top\text{-}level\text{-}cont} : \mathit{Expcont}$

$\quad \mathit{top\text{-}level\text{-}cont} = \lambda e \,.\, (\text{in}\mathit{Expressible\text{-}value}\ e)$

Give lambda expressions that define the following auxiliary functions that work with those given above.

i. $\mathit{null\text{-}env} : \mathit{Environment}$

ii. $\mathit{error\text{-}cont} : \mathit{ANSWER}$

(c) (2 points) Consider the following valuation function signatures.

$\mathcal{P}$: Program $\rightarrow$ *ANSWER*
$\mathcal{D}$: Program $\rightarrow$ *Environment* $\rightarrow$ *Expcont* $\rightarrow$ *ANSWER*
$\mathcal{E}$: Expression $\rightarrow$ *Environment* $\rightarrow$ *Expcont* $\rightarrow$ *ANSWER*
$\mathcal{F}$: Function-abstract $\rightarrow$ *Environment* $\rightarrow$ *Procedure*
$\mathcal{O}$: Operator-for-Built-in $\rightarrow$ *Nat* $\rightarrow$ *Nat* $\rightarrow$ *Value*

Given that the clause for the denotation of an expression that is a literal is:

$\mathcal{E}[\![L]\!] = \lambda u \,.\, \lambda k \,.\, (k\ (\mathcal{L}[\![L]\!]))$

what is the signature of $\mathcal{L}$? How can you tell? (Note, the correct answer may be different than was correct for the exam.)

(d) Consider the following valuation function clauses, some of which are missing or incomplete. (Some are also different than the exam.)

$\mathcal{P}[\![P]\!] = \mathcal{D}[\![P]\!]$ *null-env top-level-cont*

$\mathcal{D}[\![E]\!] = \mathcal{E}[\![E]\!]$

$\mathcal{E}[\![L]\!] = \lambda u \,.\, \lambda k \,.\, (k\ (\mathcal{L}[\![L]\!]))$

$\mathcal{E}[\![I]\!] = \lambda u \,.\, \lambda k \,.$
  cases (*lookup* u $[\![I]\!]$) **of**
  is*Denotable-value*(d) $\rightarrow$ **cases** d **of**
    is*Value*(v) $\rightarrow$ (k v)
    else *error-cont*
    end
  is*Unbound*() $\rightarrow$ *error-cont*
  end

$\mathcal{E}[\![(O\ \ E_1\ \ E_2)]\!] = \lambda u \,.\, \lambda k \,.$
  $\mathcal{E}[\![E_1]\!]u(\lambda e_1 \,.$ **cases** e_1 **of**
    is*Nat*(n_1) $\rightarrow$ $\mathcal{E}[\![E_2]\!]u(\lambda e_2 \,.$ **cases** e_2 **of**
      is*Nat*(n_2) $\rightarrow$ $(k\ ((\mathcal{O}[\![O]\!])n_1 n_2))$
      else *error-cont*
      end)
    else *error-cont*
    end)

i. (5 points) The clause for the denotation of the `if` expression is missing. What should it be?

ii. (10 points) The clauses for the denotations of combinations of the form `(`I E`)` and of the form `(`F E`)` are missing from the above. What should they be?

iii. (3 points) The clause for the denotation of $\mathcal{D}[\![$`fun` I_1 `(`I_2`)` `=` E `;` $P]\!]$ is missing above. It should have the form

$$\mathcal{D}[\![\texttt{fun}\ I_1\ \texttt{(}I_2\texttt{)}\ \texttt{=}\ E\ \texttt{;}\ P]\!] = \lambda u \,.\, \lambda k \,.$$
$$\qquad \mathcal{D}[\![P]\!]\ \ldots\ k$$

where the missing stuff (...) should be an environment that extends the environment u with the denotation of the (nonrecursive) function named I_1. This function takes an argument named I_2 and its body is the expression E. Recall that arguments are passed by value in this language. Complete this clause by stating what should replace ....