6.821 Programming Languages Handout
Fall 2002

MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Compvter Science

Problem Set 3

Problem 1: Domains

a. Let D_1 and D_2 be CPOs.

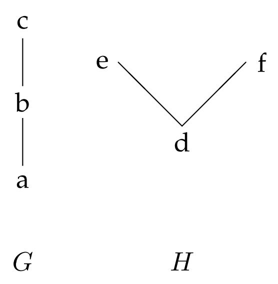
(i) Prove that any continuous function from D_1 to D_2 is monotonic.

(ii) The *componentwise ordering* of $D_1 \times D_2$ is the relation $\sqsubseteq_{D_1 \times D_2}$ defined by:

$$\langle d_1, d_2 \rangle \sqsubseteq_{D_1 \times D_2} \langle d_1', d_2' \rangle \quad \text{if and only if} \quad d_1 \sqsubseteq_{D_1} d_1' \quad \text{and} \quad d_2 \sqsubseteq_{D_2} d_2'$$

Prove that $D_1 \times D_2$ is a CPO under the componentwise ordering.

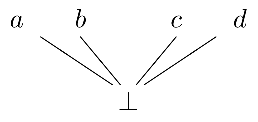
b. Let G and H be the following CPOs:



(i) Draw a Hasse diagram for the CPO $(G \times H)$.

(ii) Draw a Hasse diagram for the CPO $(H \to G)$. This is the set of continuous functions from H to G, ordered as defined on page 195 of the course notes, "Function Domains". You should write a function from H to G in an abbreviated form, as suggested by the notes. For example, the function with graph $\{\langle \texttt{d}, \texttt{a}\rangle, \langle \texttt{e}, \texttt{c}\rangle, \langle \texttt{f}, \texttt{b}\rangle\}$ can be written $\langle \texttt{a}, \texttt{c}, \texttt{b}\rangle$.

c. Let D be the following domain:



Let T be the set of total functions from D to D, let M be the set of monotonic functions from D to D, and let C be the set of continuous functions from D to D.

(i) What is the size of T?

(ii) What is the size of M? Give the graph of a function in $(T - M)$, if any.

(iii) What is the size of C? Give the graph of a function in $(M - C)$, if any.

d. Let D be a pointed CPO, and let $(D \to D)$ be the CPO of continuous functions from D to D, ordered pointwise (this is the ordering defined on page 123 of the course notes, "Function Domains").

   Define a function Y from $(D \to D)$ to D as follows:

$$Y(f) = \underset{n\geq 0}{\quad} f^n(\bot_D)$$

   Prove that Y is a continuous function.

   Addendum:

   You may use the following lemmas in your proof.

   Lemma 1: Let D be a pointed CPO, let n be any non-negative integer, and let F be a chain of functions in $(D \to D)$. Then:

$$(\underset{f\in F}{\quad} f)^n(\bot_D) = \underset{f\in F}{\quad} f^n(\bot_D)$$

   Lemma 2: Let D be a pointed CPO and let F be a chain of functions in $(D \to D)$. Then:

$$\underset{n\geq 0}{\quad}(\underset{f\in F}{\quad} f^n(\bot_D)) = \underset{f\in F}{\quad}(\underset{n\geq 0}{\quad} f^n(\bot_D))$$

Problem 2: (Desugaring)

Do exercise 6.3 (on page 162).

Addendum: Part (c) asks for a closed form solution for the number of `recs` in a desugaring of a `letrec` with n bindings. Instead, you should write a *recurrence equation* for the number of `recs` in a desugaring of a `letrec` with n bindings. In particular, you do *not* need to write a closed form solution for this recurrence equation.

Problem 3: (Substitution)

Do exercise 6.12 (on page 187).

Problem 4: (Operational semantics of FLK)

Louis Reasoner has an idea for a new FLK command, (`terminate` E_1 E_2). If either E_1 or E_2 terminates with a value or an error, (`terminate` E_1 E_2) will also terminate with a value or an error. In evaluating `terminate`, we run one expression for one step, then run the other expression for one step, and so on. Louis initially worked out a few examples of how his new construct would work:

`(terminate 1 2)` $\Rightarrow$ *1* **or** *2* **(implementation dependent)**
`(terminate 2 (rec x x))` $\Rightarrow$ *2*
`(terminate 1 (call 3 0))` $\Rightarrow$ *1* **or** *error: can't apply non-procedure* **(implementation dependent)**
`(terminate (rec x x) (/ 3 0))` $\Rightarrow$ *error: divide by zero*
`(terminate (rec x x) (rec x x))` $\Rightarrow$ $\bot$

Louis is really excited about the `terminate` construct. His old implementation of FLK required him to reboot any time his program ran into an infinite loop. Although he hasn't solved the halting problem, now he can guarantee not to have to reboot (excepting, of course, when his new-fangled operating system crashes) by testing his programs with terminate and his new `(timer` N`)` construct.

Louis defined the following transition rule(s) for `timer`:

$$\begin{array}{c} \texttt{(timer } N_1\texttt{)} \Rightarrow \texttt{(timer } N_2\texttt{)} \\ \text{where } N_1 > 1 \\ \text{and } N_2 = N_1 - 1 \end{array} \qquad [\textit{timer-countdown}]$$

$$\texttt{(timer 1)} \Rightarrow \#u \qquad [\textit{timer}]$$

Louis can now use the `terminate` construct to run his program *might-go-infinite* for exactly 1000 steps (where we consider each transition to be one step). The following expression will return the result of *might-go-infinite* if it completes in under 1000 steps, otherwise it returns `#u`.

```
(terminate (timer 1000) might-go-infinite)
```

Unfortunately, Louis set off for Hawaii before he was able to extend the FL Operational Semantics to include `terminate`. In his absence, you are asked to finish it up.

a. Give the transition rules for `terminate`.

b. Are your rules confluent?

c. Show how the following expression would be evaluated using the rules above:

```
(terminate (call (proc x (primop + x 2)) 5)
           (if (> 3 4)
               (rec x x)
               (proc y 1)))
```