What to turn in

Turn in the assignment at the beginning of class on the due date.

1. Continuity (40 pts)

   (a) (Winskel 8.6) Exactly what functions from a pointed CPO to a discrete CPO are continuous?

   (b) Show that the *strict* function,

   $$\mathit{strict} = \lambda f \in \Sigma \to \Sigma_\perp.\ \lambda \bar{\sigma} \in \Sigma_\perp.\ \mathsf{if}\ \bar{\sigma} = \perp\ \mathsf{then}\ \perp\ \mathsf{else}\ f(\bar{\sigma})$$

   is continuous.

   (c) (from Winskel 8.12) Operations on a set S can be extended to the lifted domain $S_\perp$. For example, the or-operator $\vee : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ can be extended to $\vee_\perp : \mathbb{B}_\perp \times \mathbb{B}_\perp \to \mathbb{B}_\perp$ by taking

   $$x_1 \vee_\perp x_2 = (\mathsf{let}\ b_1 = x_1, b_2 = x_2\ \mathsf{in}\ \lfloor b_1 \vee b_2 \rfloor)$$

   This extension is *strict* since if either x_1 or x_2 is $\perp$, then so is $x_1 \vee_\perp x_2$. This extension is not the only possible extension of $\vee$, however. Describe in the form of "truth tables" all continuous extensions of the boolean or-operator $\vee$. Show that the functions you define are continuous.

   (d) A set S is *isomorphic* to another set S' if there is a one-to-one function mapping from S to S'. Similarly, a domain D is isomorphic to a domain D' if there is a *continuous* one-to-one function mapping D to D'. That is, the two domains must have not only corresponding elements but also corresponding structure. Because the function is continuous, it preserves not only ordering but also suprema.

      i. Show that the domains $D \to E$ and $\{f \in D_\perp \to E_\perp \mid f(\perp) = \perp\}$ (both ordered pointwise) are isomorphic for any CPO's D and E.

      ii. Show that the domains $D \times E \to F$ and $D \to E \to F$ are isomorphic for any CPO's D, E, F.

2. Approximation (20 pts)

   An element x of a CPO *approximates* another element y, written $x \ll y$, if all chains z_n whose LUB is at least y contain an element that is at least x:

   $$y \sqsubseteq \bigsqcup_{n \in \omega} z_n \implies \exists n \in \omega . x \sqsubseteq z_n$$

   An element of a CPO is *finite* (or *compact*) if it approximates itself.

   (a) Show that $x \ll y \implies x \sqsubseteq y$.

   What are the finite elements of these domains?

   (b) natural numbers ω with discrete ordering

   (c) $\omega \cup \{\infty\}$ with $\leq$ ordering $(\forall n . n \leq \infty)$

   (d) $\mathbb{Z} \to \mathbb{Z}$ with pointwise ordering

   (e) $\mathbb{Z} \to \mathbb{Z}_\perp$ with pointwise ordering

3. Lazy uF and letrec (40 pts)

   In class (Friday) we defined a denotational semantics for the uF language, which has eager evaluation. We could as easily have written a denotational semantics for a lazy version of uF, in which let expressions, arguments to functions, and the two cells of pairs are all evaluated lazily. For example, given a divergent expression Ω, the following expressions should evaluate to 0 rather than diverging:

- left $\langle 0, \Omega \rangle$
- $(\lambda$ x $0)\ \Omega$
- let x $= \Omega$ in 0

In this problem you will write the denotational semantics for lazy uF in either direct style or in continuation-passing style (your choice). Justify any language design decisions that you have to make along the way.

(a) Write the domain equations for lazy uF.

(b) Define a semantic function $\mathcal{C}[\![\cdot]\!]$ that gives the meaning of a lazy uF expression.

uF has the simple recursion construct rec that permits construction of recursive functions. Many languages allow the construction of mutually recursive functions, like the REC language that we defined in class. In a lazy language, there is the potential to define recursive *data structures* as well. Suppose we want both of these capabilities in lazy uF, and extend the language with a corresponding letrec expression:

$$e ::= \ldots \mid \mathsf{letrec}\ x_1 = e_1, \ldots, x_n = e_n\ \mathsf{in}\ e_0$$

Because arbitrary expressions can appear in a letrec expression (unlike in ML), we can use letrec to define recursive data structures, such as infinite lists of the natural numbers. For example, the following evaluates to 4:

```
letrec
  inclist = (λ x ⟨(left x) + 1, inclist (right x)⟩),
  nats = ⟨0, inclist nats⟩
  evens = ⟨0, inclist odds⟩,
  odds = inclist evens,
in
  left (right (right evens))
```

The same approach can be used to produce other interesting infinite lists, such as a lazily-evaluated list of the primes.

(c) Define $\mathcal{C}[\![\mathsf{letrec}\ x_1 = e_1, \ldots, x_n = e_n\ \mathsf{in}\ e_0]\!]$. Justify the correctness of any use of *fix* that appears in your definition.

(d) Show that your language semantics gives the intended meaning for the expression letrec ones=⟨1, ones⟩ in ones.

(e) What goes wrong if we try to add this expression form to standard (eager) uF?