Homework 5

This homework is due Friday, Mar. 23 (in class).

1. (4 points) Consider a function $f : A \to B$ where A and B are pointed cpos.
   (a) If f is strict, is it necessarily monotonic?
   (b) If f is monotonic, is it necessarily strict?
2. (5 points) From the domain equations and the signature of $\mathcal{E}$ in the denotational semantics of the language SL, it is possible to infer that expression evaluation always terminates. One need not inspect the abstract syntax or the valuation functions to determine this fact. *Briefly* explain why.
3. (corrected version) A general framework for program analysis and optimization is provided by *abstract interpretation*, which consists of interpreting a program before run-time, using a different domain than the one used in the language's semantics. As a simple example, one might interpret integer expressions in the domain

$$Sign = (\{\top, -, 0, +, \bot\}, \sqsubseteq),$$

where $\sqsubseteq$ is defined so that $\bot$ is the least element of *Sign*, $\top$ is the greatest element, and the elements $-$, $+$, and 0 and not related to each other by $\sqsubseteq$.

Let the function *abs* from the domain $Int_\bot$ to *Sign* be defined so that it maps negative numbers to $-$, zero to 0, positive numbers to $+$, and $\bot$ to $\bot$. For example, $abs(-3) = -$.

The operations on the domain *Sign* are abstractions of the integer operations. Let *abs* also represent the map from the operations on *Int* to the corresponding operations on *Sign*. For example, the abstract operation corresponding to integer multiplication ($\times$) is $abs(\times)$. For correctness, it is required that for all integer operations f, and for all integers x and y,

$$abs(f(x, y)) \sqsubseteq (abs(f))(abs(x), abs(y)). \qquad (1)$$

For example, $abs(\times)$ is defined by the following Caley table:

$abs(\times)$	$-$	0	$+$	$\bot$	$\top$
$-$	$+$	0	$-$	$\bot$	$\top$
0	0	0	0	$\bot$	0
$+$	$-$	0	$+$	$\bot$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\top$	$\top$	0	$\top$	$\bot$	$\top$

The table can be derived from Formula 1 by always picking the least choice for each entry. In particular to derive the entries involving $\top$, one must consider that an argument of $\top$ is an abstraction of all elements (or any element) of the the domain $Int_\bot$. Thus since a negative number times some other random element can have any sign, the appropriate entry is $\top$. Think of $\bot$ as nontermination. Then nontermination times any element of $Int_\bot$ gives nontermination, so the proper entry in the table is as above. The only other funny entry is 0 times $\top$, which is 0, not $\top$. This is because 0 times anything but $\perp$ is 0, and 0 times $\perp$ is $\perp$, which approximates 0.

In general there may be many possible rules for $abs(f)$ that satisfy Formula 1. The *best* choice is the least such function on *Sign*, under the usual domain ordering for $Sign \rightarrow Sign$. For example, the function that maps all pairs of elements of *Sign* to $\top$ is greater than the function $abs(\times)$ described above, although it satisfies Formula 1 for $\times$.

(a) (4 points) What is the result of $abs(\times)(abs(3), abs(-4))$?

(b) (4 points) What is the best choice for $abs(\div)$, that is for division? Assume that division by zero returns $\perp$.

(c) (6 points) What is the best choice for the abstract operation $abs(+)$, that is for addition?

(d) (4 points) What is the result of $abs(+)(abs(3), abs(-4))$?

(e) (10 points) In general a expression may involve calls to user-defined functions. Briefly describe a strategy for abstract interpretation of recursive definitions of user-defined functions (such as factorial) which yields the most information about the result, but which guarantees termination of the abstract interpretation.

4. (20 points) Algol 60 provides both pass by name and pass by value in the same language. This problem considers an extension to FL that will allow both parameter passing mechanisms to coexist.

   Extend the abstract syntax of FL by adding the `strict` construct:

   $$\text{E} ::= \cdots \mid (\texttt{strict}\ \text{I E})$$

   Like `proc`, `strict` creates a procedure; whereas `proc` creates a pass by name procedure, `strict` creates a procedure that takes its arguments by value.

   Beginning with the denotational semantics for pass by name FL, make extensions as needed to define the `strict` construct. Turn in a list of all modified or changed valuation clauses as well as any changes you make to the domain equations.

5. (20 points) Modify the interpreter in `~leavens/cs541public/fx/fl.fx` on snowhite to implement the `strict` construct. Refer to the introduction to mini-fx handout for the mini-fx programming. It is essentially like Scheme, with the addition of sums and products. (Don't worry about the commented-out definitions in the file.)

   Turn in a listing of the procedures that you modified (not the entire interpreter) as well as mailing these procedures to `coleman` on snowhite.

   (Hint: if you feel you must change more than a few procedures, you're probably doing something wrong.)