Homework Assignment #2

Due Wednesday 2/8, at the start of lecture

1. Given a set S_1 and a lattice $D_2=(S_2,\leq_2)$, one can construct the lattice $S_1\rightarrow D_2=(S_\rightarrow,\leq_\rightarrow)$ of all **total** functions from S_1 to S_2. A element of $S_1\rightarrow D_2$ is a set of pairs (a,b) where $a\in S_1$, $b\in S_2$, and each element of S_1 appears in the set exactly once. One element f of $S_1\rightarrow D_2$ is ordered less than or equal to another element g of $S_1\rightarrow D_2$ iff, for all elements a of S_1, $f(a) \leq_2 g(a)$, where $f(a)=b$ iff $(a,b)\in f$.

   a. Define formally the lattice $S_1\rightarrow D_2$ by defining $S_\rightarrow$ and $\leq_\rightarrow$ in terms of S_1, S_2, and $\leq_2$.

   b. Define $\cap_\rightarrow$, $\top_\rightarrow$, $\bot_\rightarrow$, and the height of $S_1\rightarrow D_2$ in terms of S_1, S_2, $\leq_2$, $\cap_2$, $\top_2$, $\bot_2$, and the height of D_2. (Note that these other values are completely defined by $S_\rightarrow$ and $\leq_\rightarrow$. so technically it's redundant to specify them, but it's useful and conventional to do so, and is a partial check on whether $S_\rightarrow$ and $\leq_\rightarrow$.actually form a lattice.)

   c. Formally define helper functions `lookup`: $S_\rightarrow \times S_1 \rightarrow S_2$ (which takes a function and an element of the function's domain and returns the corresponding element of the function's range), and `update`: $S_\rightarrow \times S_1 \times S_2 \rightarrow S_\rightarrow$ (which takes a function, a domain element, and a range element, and returns a new function that maps the domain element to the range element and maps all other elements the same way as the input function).

   d. Use this $\rightarrow$ lattice constructor to specify an appropriate lattice for reaching constants. Explain any relationship between this lattice and the "best" lattice defined in lecture.

   e. Formally define the flow function $CP_{X := Y + Z}$: $S_\rightarrow \rightarrow S_\rightarrow$. Make use of your `lookup` and `update` helpers.

2. a. Perform the may-point-to analysis described in class (using simple allocation-site summary nodes) on the following code fragment:

```
{
decl a, b, c, d, e;
a := &b;
if (...) {
   *a := &d;
} else {
   b := &c;
}
e := a;
do {
   decl t;
   t := new (void*);
   *t := *e;
   e := t;
} while (...);
```

```
decl f, g;
*a = &f;
*e = &g;
}
```

(In this intermediate language, variables are untyped and any variable may hold a pointer to any other variable, `decl` statements introduce new uninitialized local variables that are in scope from the point of declaration until the end of their brace-delimited scope, and `new (void*)` allocates new heap memory big enough to hold a pointer. No variables are in scope at the start of this fragment.)

Draw the control flow graph for this fragment, where each node is an individual statement, merge node, or conditional branch node (with unknown branch condition). Annotate each edge in the CFG with a may-point-to graph whose nodes are all local variables in scope. In the case of iterative analysis, clearly indicate which edges are added in each iteration.

b. When revisiting a given program point during iterative analysis, why are may-point-to edges only added, never removed?

c. Why is it important to treat assignments to allocation-site summary nodes differently than other nodes?