Homework 2

Due **Wednesday, February 11**, at the start of class

Turn in a hard copy of your answers. The usual rules about collaboration, not finding solutions elsewhere, etc., apply.

1 Meet is monotonic

Prove that $\sqcap$ is monotonic. In other words, given a lattice $(D, \sqsubseteq, \bot, \top, \sqcap)$, show that $\forall a, b, c, d \in D. a \sqsubseteq b \wedge c \sqsubseteq d \Rightarrow (a \sqcap c \sqsubseteq b \sqcap d)$. Your proof should be rigorous, in that every step needs to be justified. The only facts that you can use are:

- the definition of $\sqcap$: $a \sqcap b$ is the element $c \in D$ such that $c \sqsubseteq a \wedge c \sqsubseteq b \wedge \forall d.[(d \sqsubseteq a \wedge d \sqsubseteq b) \Rightarrow d \sqsubseteq c]$.
- transitivity: $a \sqsubseteq b$ and $b \sqsubseteq c$ implies $a \sqsubseteq c$
- commutativity: $a \sqcap b = b \sqcap a$

2 Constant prop flow function is monotonic

Prove that the following constant prop flow function is monotonic:

$$CP_{x:=y+z}(in) = in - \{(x \to *)\} \cup \{(x \to c_3) \mid (y \to c_1) \in in \wedge (z \to c_2) \in in \wedge c_3 = c_1 + c_2\}$$

In other words, prove that, if $in_1 \subseteq in_2$, then $CP_{x:=y+z}(in_1) \subseteq CP_{x:=y+z}(in_2)$. Your proof should be rigorous. You'll need to use the following facts (if you don't, then chances are that your proof is not rigorous enough):

- $(A \subseteq B) \iff \forall x.[(x \in A) \Rightarrow (x \in B)]$
- $x \in (A \cup B) \iff (x \in A) \vee (x \in B)$
- $x \in (A - B) \iff (x \in A) \wedge (x \notin B)$

3 Map lattice constructor

Given a set D_1 and a lattice $(D_2, \sqsubseteq_2, \bot_2, \top_2, \sqcap_2)$, define the domain $D_1 \mapsto D_2$, which represents the lattice of maps from set D_1 to domain D_2. In particular, $D_1 \mapsto D_2 = (D, \sqsubseteq, \bot, \top, \sqcap)$ and you need to define $D, \sqsubseteq, \bot, \top$ and $\sqcap$. An element of your domain D should be a set of pairs restricted in such a way so that it represents a total function. (A function f is a set of pairs such that for any x there is at most one y such that $(x, y) \in f$. A total function f is a function that has the property that for any x in the domain of f, there is an element $(x, y) \in f$.)

Show how to use this lattice constructor to define a domain D_{cp} for constant prop.

Write the flow function $CP_{x:=y+z}$ using the D_{cp} domain. It may be helpful to define some map domain helper functions, such as lookup and update.

4 SSA

Put the following program in SSA form (you may draw a control flow graph to illustrate your solution):

```
x := 0;
do {
   x := x + 1;
   z := x;
   y := 0;
   if (...) {
      y := 1;
   }
   w := y + z;
} while (...);
print(x, y, z, w);
```

5 Constant prop over def-use chains

Give an algorithm for constant propagation that exploits def/use chains to work faster than the propagation-based algorithm presented in class. What is the time complexity of your algorithm, assuming def/use chains are already constructed? How, if at all, would converting the program to SSA form before constructing def/use chains help your analysis?

6 Liveness analysis over def-use chains

Give an algorithm for dead assignment elimination that exploits def/use chains to work faster than the propagation-based algorithm that used live variables analysis presented in class. Your algorithm should not miss any optimization opportunities found by the best live variables-based algorithm presented in class. What is the time complexity of your algorithm, assuming def/use chains are already constructed? How, if at all, would converting the program to SSA form before constructing def/use chains help your analysis?

7 Pointer analysis

Perform the pointer analysis defined in class (using simple allocation-site summary nodes as on slide 93) on the following program:

```
struct T { field f } // struct T has a field f
T g;                 // global of type T
main() {
  x := &y;
  if (...) {
    do {
      if (...) {
        z := &y;
        a := new T;
      }
      else {
        z := &z;
        a := &g;
      }
      a->f := *z;
      *z := &a;
    } while (...)
  }
  else {
    t := &y;
    *x := t;
  }
```

Draw the control flow graph, and draw the points-to graphs representing the dataflow information computed at each program point. If there is iteration, identify clearly which edges and nodes are added at each iteration.

8 Correctness of pointer analysis

8.1 Abstraction relation

Define the α_{pt} relation for a may-points-to analysis assuming that the domain of the analysis is $Pow(Var \times Var)$ (no dynamically allocated memory or other summaries). You can use a model of concrete memories that has the same domain, except that it must be a partial function: any variable points to at most one target variable. (We'll keep things as simple as possible by assuming that all variables hold pointers, or null; no other values are in our memories.)

8.2 Local soundness [Extra Credit]

Consider the following may-points-to flow function for statements of the form $x := y$:

$$MAY\text{-}PT_{x:=y}(in) = in - \{(x, *)\} \cup \{(x, v) \mid (y, v) \in in\}$$

Also, consider the following definition of the concrete semantics of statements of the form $x := y$ (ignoring the in and out program points):

$$mem_{in} \rightarrow_{x:=y} mem_{in}[x \mapsto mem_{in}(y)]$$

Show that the abstract flow function $MAY\text{-}PT_{x:=y}$ is sound with respect to α_{pt} and the concrete flow function $\rightarrow_{x:=y}$. In other words, show that if $mem_{in} \rightarrow_{x:=y} mem_{out}$, $(mem_{in}, d_{in}) \in \alpha_{pt}$, and $MAY\text{-}PT_{x:=y}(d_{in}) = d_{out}$, then $(mem_{out}, d_{out}) \in \alpha_{pt}$.