UNIVERSITY WISCONSIN MADISON, COMPUTER SCIENCE DEPARTMENT

CS704 Assignment 5

Yan Zhai

May 10, 2013

PROBLEM 1

PART(A)

This is a galois insertion:

1. we try to prove $\alpha(\gamma(a)) = a$ for all $a \in D_1$
2. when $a = \bot$, $\gamma(\bot) = \emptyset$, $\alpha(\emptyset) = \bot$, the conclusion apparently holds
3. for other values of a, it's enough to show the case of just one variable v.
   - if $a = (v \to c)$, and c is a constant, then $\gamma(a) = (v \to c)$, and $\alpha((v \to c)) = (v \to c)$. so the conclusion holds.
   - if $a = (v \to \top)$, then $\gamma(a) = (v \to N)$, here N is the top element in concrete domain, and $\alpha(v \to N) = (v \to \top)$.
4. based on above points, the proof completes.

PART(B)

- abstraction: $\emptyset$ is mapped to bottom element of abstract domain 2. For other element in concrete domain, for each variable, if all the states map it to the same integer c, then in the corresponding abstract state it's mapped to c, too. Else it's mapped to $\top$.
- concretization: any state contains mappings to $\bot$ are mapped to $\emptyset$. For those without $\bot$ mapping, it is same as the concretization function in abstract domain 1.

proof of being galois connection:

- for all $a \in A_2$, if a contains $\bot$ mapping (including bottom element), $\gamma(a) = \emptyset$, and $\alpha(\emptyset) = (v_1 \to \bot, v_2 \to \bot ...) \leq a$.
- for all other $a \in A_2$, it works the same as the two functions in abstract domain1. So it's galios connection.

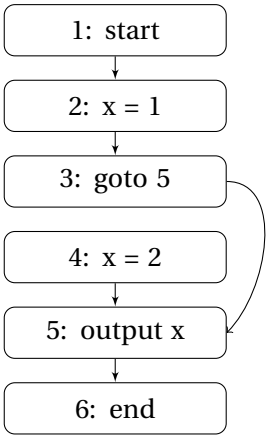
PART(C)

The second pair of abstraction and concretization functions are not galois insertion: $\alpha(\gamma(v_1 \to 1, v_2 \to \bot)) = \alpha(\emptyset) = (v_1 \to \bot, v_2 \to \bot)$. So it's not galois insertion.

PROBLEM 2

PART(A)

We reuse the abstract domains and functions in Problem1. Let's check this control flow graph:



The control flow graph can never reach node 4. When we use abstract domain 1 and its insertion functions, node 4 always contain state $\bot$, thus when at node 5 we calcuate the previous abstract states, we won't introduce new state other than those from node 3. However, when we use abstract domain 2, node 4 has state $(x \to \bot)$, and use the control transfer function, we will produce a state $(x \to 2)$, and combine with other state, it will become $(x \to \top)$, which can never happen!

Based on the above observation, the insertion functions would be better since they can not produce junk states.

PART(B)

For abstract domain 1, we just make the transfer function for the branch (x < 3) to be

$$\lambda s.\text{if lookup(s, x)} < 3 \text{ then } s \text{ else } \bot$$

For the other branch just reverse the inequality.

For abstract domain 2, we just make the transfer function for the branch (x < 3) to be:

$$\lambda s.\text{if lookup(s, x)} < 3 \text{ then } s \text{ else } \{x \to \bot, y \to \bot, z \to \bot\}.$$

For the other branch just reverse the inequality.

PART(C)

For above case, it still has the problem as in Part(a). The reason is even if a branch is dead, it can still have control transfer edge to following nodes. If there are three assignments like x=1,y=2,z=3 inside the dead branch, then for abstract domain 2 it can produce junk states $\{x \to 1, y \to 2, z \to 3\}$. If before the dead branch, x is mapped to 2, then at these nodes the final abstract states will map x to $\top$. On the other hand, this will not happen in abstract domain 1. This is because dead branch can only produce state $\bot$ at any time if it's unreachable. And this will avoid introducing tainted states, and produce a better constant propagation result.