CS243 HW 1 Solutions

Winter 2017

Note: these solutions are slightly more detailed than we would expect you to hand in so that you can learn from any questions you missed.

Problem 1

1. Component-wise minimum for tuples of integers is a meet operator, it is idempotent, commutative and associative.
2. Addition over complex numbers is not a meet operator as it is not idempotent.
3. Multiplication over complex numbers is not a meet operator as it is not idempotent.
4. Arithmetic mean over complex numbers is not a meet operator as it is not associative.
5. AND over booleans is a meet operator, it is idempotent, commutative and associative.

Problem 2

1. False, MFP solution is in general not equal to MOP when $\wedge$ is non-distributive over the transfer functions.
2. False, there can be multiple fixed point solutions, such as can be constructed by adding spurious live variables in a loop for live variable analysis.
3. False. A bottom element b is one such that $\forall x. x \wedge b = b$. If there were two bottom elements b_1 and b_2, then $b_1 \wedge b_2 = b_1$ by b_1 being a bottom, and $b_1 \wedge b_2 = b_2$ by b_2 being a bottom, so they must be the same element.
4. False, only *bounded* semi-lattices have a top element.

Problem 3

1. Yes, the algorithm gives a safe answer for all flow graphs because the incorrect initialization can only drive the meet for the IN's down the lattice.
2. N/A
3. No, the algorithm will not give MOP for all flow graphs, because it will not give the maximum fixed point for some graphs.
4. Yes, it will compute the MOP on any acyclic flow graph.

Problem 4

Compute both reaching definitions as well as liveness for each variable. A program point P is in the live range of a definition $D : x = \dots$ if and only if D reaches point P **and** x is live at the point P.

Problem 5

1.

Domain	Sets of expressions
Direction	Forward
Transfer Function	$f_b(x) = (\text{Computed}[b] \cup x) - \text{E}_{\text{Kill}}[b]$
$\wedge$	$\cap$
Boundary	$\text{out[entry]} = \emptyset$
Initialization	$\text{out[b]} = \{\text{all expressions}\}$

Here Computed$[b]$ is the expression on the right hand side of statement b. $E_{\text{kill}}[b]$ is all expressions that involve the variable defined on the left hand side of b. Note that a statement like `a = a + c` generates {`a + c`} but then immediately kills it (along with all other `a` expressions). The transfer function for a non-assignment (like say a function call) is the identity.

2. (a) 10001

   (b) 10000. Only the final evaluation can be eliminated because the expression is not available in the loop on the first iteration.