Winter 2007 Midterm Solutions

Problem 1: True/False

(a) (3 points) If a node n_1 dominates a node n_2, then n_1 is always visited before n_2 in a depth-first search.
TRUE. By the definition of domination, all paths that lead to n_2 go through n_1 first. So, a depth-first search must reach n_1 first.
- OR -
FALSE. A post-order depth-first search visits all children of a node before visiting the node itself. By the definition of domination, n_2 must be visited first, since it is a child.

(b) (3 points) If a node n_1 dominates a node n_2, n_1 is always visited before n_2 in a reverse post ordering.
TRUE. Justification is essentially identical to the "False" answer for 1(a) -- if a post-ordering must list n_2 first, then a reverse post-ordering must list n_1 first.

(c) (3 points) Given a monotone data flow framework, had all the interior points of the data flow solver been initialized with the "bottom" of the semi-lattice, the answer at every point in every program would have been "bottom."
FALSE. Even though any element met with bottom is itself bottom, this makes no assumption about the results of the transfer function. Suppose the transfer function returns a single constant value (not bottom). This is definitely monotone (no matter the values of x and y, $f(x) \leq f(y)$), and the values once the solution stabilizes will definitely not all be bottom.

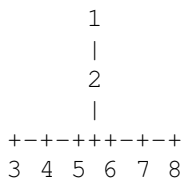
(d) (4 points) Given a machine with 5 registers, Chaitin's register coloring algorithm (the one presented in class) will succeed in coloring any program that has no more than 4 active live ranges at any point in the program.
FALSE. Unfortunately, the number of live ranges at any given point has nothing to do with the interference graph. Consider the following set of merged live ranges:

#	A	B	C	D	E	F
1	X	X	X	X		
2	X	X		X	X	
3	X			X	X	X
4				X	X	X
5			X		X	X
6		X	X			X
7	X	X	X			

This graph only has four live ranges at any given point, but every variable interferes with every other one, and so this cannot be 5-colored.

Problem 2: Dominators, loops, and reducibility

(a) (3 points) Show the dominator tree for this flowgraph.
1 dominates everything, 2 dominates everything but 1, and nothing else dominates anything but itself. The dominator tree looks like this:

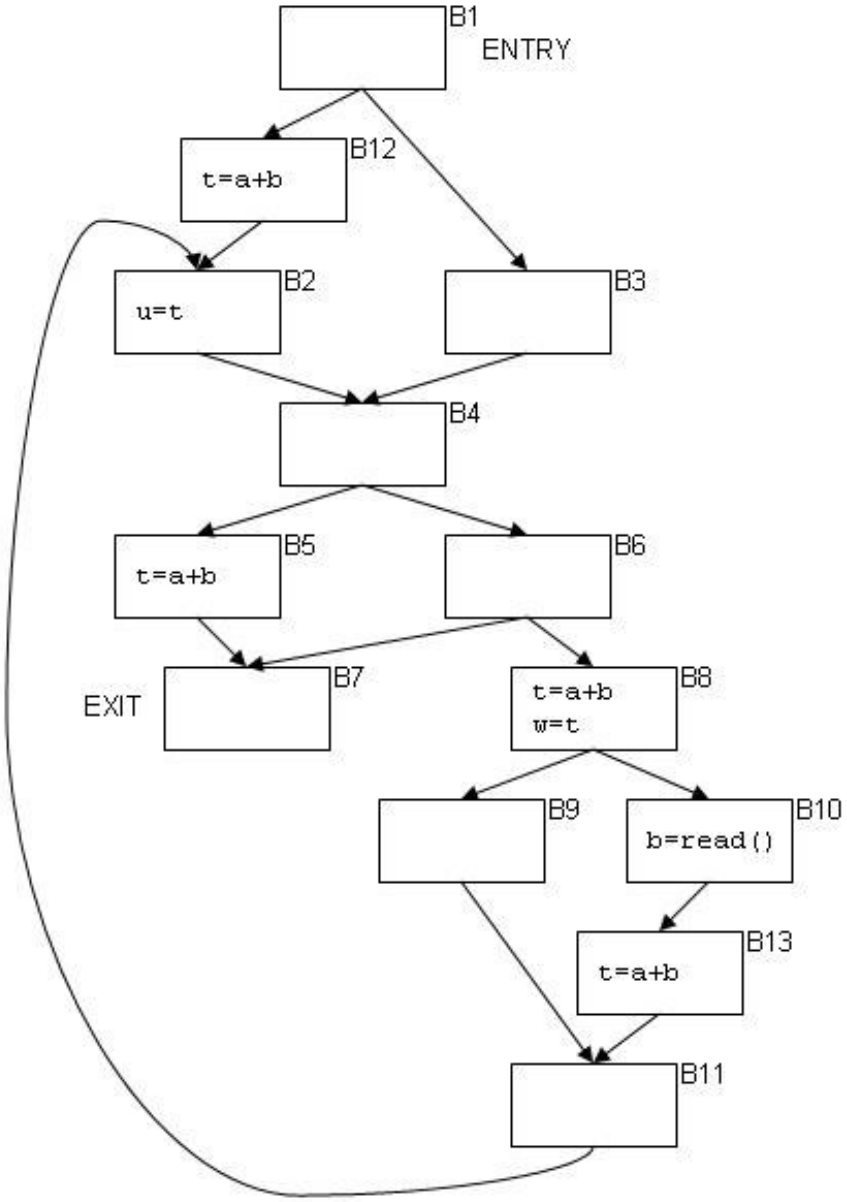


(b) (2 points) What are the back edges of this flowgraph?
The back edges are 4->1 and 6->2.

(c) (4 points) Find the natural loop associated with each of the back edges.
4->1: {1 2 3 4 6}
6->2: {2 3 4 6}

(d) (3 points) Is the flow graph reducible?
No. The edge 7->5 is a retreating edge that is not a back edge. The graph is not reducible.

Problem 3: Optimization



Problem 4: Dataflow analysis

For simplicity assume one statement per basic block. Composition is trivial. Once the dataflow problems have been solved, printing of the Warnings is done using the content of the block and the result of the analysis at the block.

I. Lock -> Lock

Note that warning I is emitted on the second LOCK.

- Forward
- {NoLock, Lock}
- NoLock -> Lock
- All points (including boundary) are NoLock.
- Transfer function = Lock if stmt is LOCK; NoLock if stmt is UNLOCK; and simply propagates the input if the stmt is something else.
- Monotone, Distributive, and convergent (monotone & finite descending chains).

Issue Warning I on any LOCK stmt that has an in value of Lock.

II. Unlock -> Unlock

Warning II, like warning I, is emitted on the second instruction. The problem is very similar to I:

- Forward
- {NoUnlock, Unlock}
- NoUnlock -> Unlock
- All points (including boundary) are NoUnlock.
- Transfer function = Unlock if stmt is UNLOCK; NoUnlock if stmt is LOCK; and simply propagates the input if the stmt is something else.
- Monotone, Distributive, and convergent (monotone & finite descending chains).

Issue Warning II on any UNLOCK stmt whose in statement is Unlock.

III. Lock -> exit

The question asks you to issue a warning "on a LOCK operation if..." . Thus it has to either be a backward problem, or it has to be a forward problem that tags each LOCK statement and treats them like reaching definitions (issuing the warning if any LOCK operation reaches the exit block).

- Backward
- {Exits, CannotExit}
- CannotExit -> Exits
- Interior points are CannotExit.

- in[exit] is Exits.
- Transfer function = CannotExit if the stmt is LOCK or UNLOCK and propagates its output otherwise.
- Monotone, Distributive, and convergent (monotone & finite descending chains).

Issue Warning III on any LOCK statement whose OUT value is Exits.

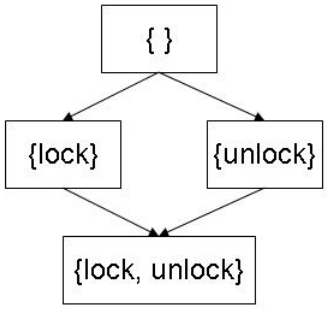
IV. entry -> Unlock

This is very similar to Warning III, except that it runs forward.

- Forward
- {Enters, CannotEnter}
- CannotEnter -> Enters
- Interior points are CannotEnter.
- out[entry] is Enters.
- Transfer function = CannotEnter if the stmt is LOCK or UNLOCK and propagates its input otherwise.
- Monotone, Distributive, and convergent (monotone & finite descending chains).

Issue Warning IV on any UNLOCK statement whose IN value is Enters.

We can actually combine the analyses for warnings I, II, and IV. The lattice will now look like this:



The analysis functions pretty much as stated before. However, Warning III still requires either a backwards analysis, or a forward analysis of an entirely different nature. You'll need at least two lattices to do it right.