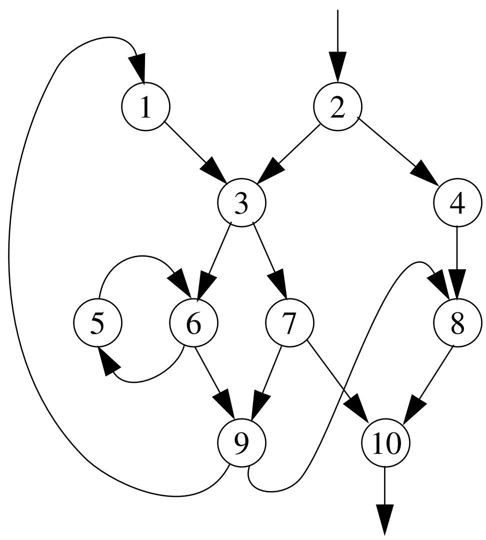
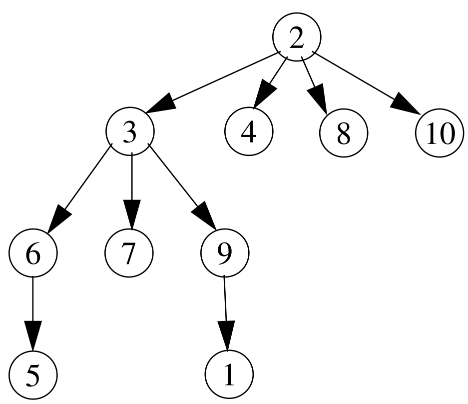
Homework Assignment #1 Sample Solutions

Due Monday 1/25, at the start of lecture

1. For the following control flow graph, for the purposes of a forward data flow analysis (such as available expressions), show the dominator tree, identify any loops, and for each loop, identify the loop head node and the loop entry, exit, and back edges. Is this graph reducible?



Dominator tree:



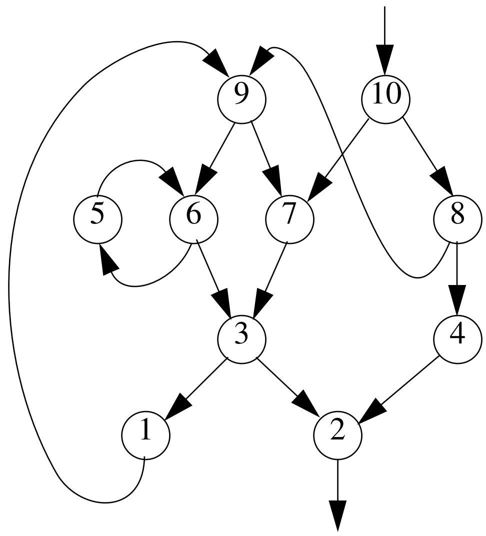
Loop 1: nodes: 1, 3, 5, 6, 7, 9; head node: 3; back edge: 1→3; loop entry edge: 2→3; loop exit edges: 7→10, 9→8

Loop 2: nodes: 5, 6; head node: 6, back edge: 5→6; loop entry edge: 3→6; loop exit edge: 6→9

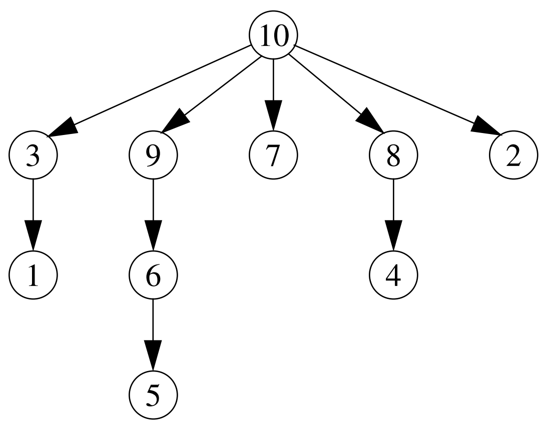
The graph is reducible.

2. Repeat problem 1 for purposes of a backwards data flow analysis (such as live variables).

Reversed control flow graph:



Dominator tree:



Loop: nodes: 5, 6; head node: 6, back edge: 5→6; loop entry edge: 9→6; loop exit edge: 6→3

(There's also a larger strongly connected component, but it has two loop head nodes, so it's not a true natural loop: nodes: 1, 3, 5, 6, 7, 9; head nodes: 7, 10; back edges (i.e., in-loop edges to a head): 1→9, 9→7; loop entry edges: 10→7, 8→9; loop exit edge: 3→2)

The graph is not reducible.

3. If a dataflow analysis can be represented as a bit-vector problem, and the KILL and GEN bit-vectors for each flow function can be expressed independently of the input bit-vector, then the composition of a series of flow functions (representing for instance the effect of analyzing all the instructions in a basic block) can represented by a single pair of summary KILL and GEN bit-vectors. Give a pair of simple inductive equations for computing the SUMMARY-KILL$_i$ and SUMMARY-GEN$_i$ bit-vectors for the 0 through *i*th instructions in a basic block from the SUMMARY-KILL$_{i-1}$ and SUMMARY-GEN$_{i-1}$ bit-vectors (if i>0) and the KILL$_i$ and GEN$_i$ bit-vectors for the flow function of the *i*th instruction.

$SUMMARY\text{-}KILL_i = SUMMARY\text{-}KILL_{i\text{-}1} \cup KILL_i$

$SUMMARY\text{-}GEN_i = SUMMARY\text{-}GEN_{i\text{-}1} - KILL_i \cup GEN_i$

where $SUMMARY\text{-}KILL_{\text{-}1}$ and $SUMMARY\text{-}GEN_{\text{-}1}$ are $\varnothing$

(Note that $SUMMARY\text{-}KILL_i = SUMMARY\text{-}KILL_{i\text{-}1} - GEN_{i\text{-}1} \cup KILL_i$ looks more correct than the above equation, but they have the same end effect.)

4. Define a data flow analysis that can be used to compute two sets of variables: those which are definitely not defined before use, and those which may not be defined before use; these sets are intended to provide extra error checking of programs. Strive for the highest quality information while maintaining safety. Use a lattice-theoretic formalism to define your analysis. Argue for the termination of your analysis. Explain how to use the information computed by your analysis to construct the two sets specified above for output to the user.

It is possible to construct either a forward or a reverse dataflow problem to answer this question. We sketch out both possibilities; either one is a sufficient answer. We'll assume that any formal parameters are initialized via explicit assignments in the flow graph.

The first approach is to perform a forward dataflow analysis computing for each program point a set of definitely defined variables and a set of possibly defined variables. After these sets are computed, we can then examine all of the uses in the program and report any errors. A variable is definitely not defined before it is used when the variable does not appear in the possibly defined variable set associated with the program point immediately preceding a statement in which the variable is used. A variable may not be defined before it is used if at the program point proceeding a statement in which it is used it is not an element of the definitely defined variable set. Note that a read through a pointer must be considered to be a use of every variable, due to the lack of more detailed pointer analysis.

To simplify the formal definition we'll define two lattices, and then use the 2-Tuple constructor to compose them into a single lattice.

Definitely Defined Variables:

Elements: *Pow(Variables), where Variables is the set of all variables in the CFG*

≤: $\subseteq$

meet: $\cap$

top: *Variables*

bottom: $\varnothing$

Flow Functions:

$F_{x := \ldots}$ $Info_{succ} = Info_{pred} \cup \{x\}$

$F_{*x := \ldots}$ $Info_{succ} = Info_{pred}$

Possibly Defined Variables:

Elements: *Pow(Variables)*

≤: ⊇

meet: ∪

top: ∅

bottom: Variables

Flow Functions:

$F_{x := \ldots}$ $Info_{succ} = Info_{pred} \cup \{x\}$

$F_{*x := \ldots}$ $Info_{succ} = Info_{pred} \cup Variables = Variables$

The flow functions are monotonic and the lattices are finite height, so the analysis will terminate. (Each individual lattice has height O(N); when we compose them into a 2-Tuple we still have a O(N) height lattice. Each flow function can run in O(N) time (or less). So, we get an $O(N^3)$ *analysis to compute the sets. The post-pass to examine the sets and produce warnings can run in* $O(N^2)$ *time (O(N) statements, O(N) work for each (O(N) for reads through a pointer).)*

Alternatively, one could define a reverse dataflow analysis that computes the set of possibly live and definitely live variables at each program point. All variables that are definitely live at the flow graph entry point are definitely used before they are defined; all variables that are possibly live at the flow graph entry point may be used before they are defined.

Just as before, we'll define two lattices and then combine them using the 2-Tuple operator:

Definitely Live Variables:

Elements: *Pow(Variables)*

≤: ⊆

meet: ∩

top: *Variables*

bottom: ∅

Example Flow Functions:

$F_{x := y + z}$ $Info_{succ} = Info_{pred} - \{x\} \cup \{y,z\}$

$F_{*x := y + z}$ $Info_{succ} = Info_{pred} - Variables \cup \{y,z\} = \{y,z\}$

$F_{x := *y + *z}$ $Info_{succ} = Info_{pred} - \{x\} \cup \emptyset = Info_{pred} - \{x\}$

$F_{*x := *y + *z}$ $Info_{succ} = Info_{pred} - Variables \cup \emptyset = \emptyset$

Possibly Live Variables:

Elements: *Pow(Variables)*

≤: ⊇

meet: ∪

top: $\emptyset$

bottom: Variables

Example Flow Functions:

$F_{x := y + z}$ $Info_{succ} = Info_{pred} - \{x\} \cup \{y,z\}$

$F_{*x := y + z}$ $Info_{succ} = Info_{pred} - \emptyset \cup \{y,z\} = Info_{pred} \cup \{y,z\}$

$F_{x := *y + *z}$ $Info_{succ} = Info_{pred} - \{x\} \cup Variables = Variables$

$F_{*x := *y + *z}$ $Info_{succ} = Info_{pred} - \emptyset \cup Variables = Variables$

Termination and complexity are identical to the forward analysis.