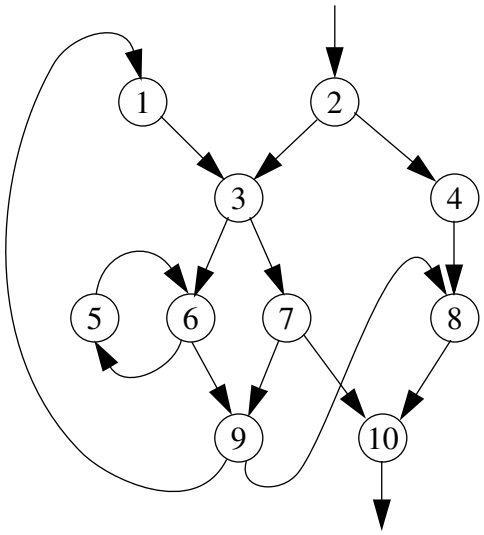
Homework Assignment #1

Due Wednesday 1/21, at the start of lecture

(For all homeworks, do not look at any previous sample solutions for any previous 501 homework assignments.)

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



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

3. Consider the case where a forward analysis's flow functions for each kind of statement are represented in terms of constant KILL and GEN bit-vectors. Thus, each flow function has the form *out* = *in* - KILL + GEN, with KILL and GEN being constants independent of *in* or *out*. The combined effect of a whole basic block of instructions can be calculated by applying each instruction's flow function in turn, starting with the in bit-vector at the start of the basic block, and computing the out vector at the end of the basic block. I.e., for instruction i of the basic block, $out_i = in_i$ - KILL_i + GEN_i, and for all instructions other than the first, $in_i = out_{i-1}$. However, in this special case where KILL and GEN do not depend on *in*, it is possible to compute a single summary flow function with constant KILL and GEN bit-vectors, of the form $out_n = in_1$ - SUMMARY-KILL + SUMMARY-GEN (where in_1 and out_n are the points before and after the basic block, respectively).

   Give a pair of simple inductive equations for computing the SUMMARY-KILL_i and SUMMARY-GEN_i bit-vectors, representing the summary KILL and GEN bit-vectors for the first i instructions of a basic block, in terms of the $\text{SUMMARY-KILL}_{i-1}$ and SUMMARY-GEN_{i-1} bit-vectors (if $i>1$) and the KILL_i and GEN_i bit-vectors for the flow function of the ith instruction, $1 \le i \le n$. SUMMARY-KILL and SUMMARY-GEN for the whole basic block are then SUMMARY-KILL_n and SUMMARY-GEN_n, respectively.

What is the chief advantage of this SUMMARY flow function over the original flow functions?

4. Consider an IR with the following grammar for its statements:

```
Stmt  ::= LHS ":=" RHS      assignment statements
        | ...               other uninteresting statements

LHS   ::= Var               assign to a variable
        | "*" Var           assign through a pointer variable

RHS   ::= Const             a constant
        | Var               a variable
        | Var Op Var        a binary operation over two variables
        | "*" Var           load through a pointer variable
```

   Define the flow functions for constant propagation, including simulated constant folding, for assignment statements in this IR. Avoid any redundancy by specifying your flow function in terms of two helper functions, one for the left-hand-side of assignments, and one for the right-hand-side of assignments. (Note that this IR strives to keep the definition of analyses simple, breaking down some complicated kinds of expressions seen in class, e.g. `*r:=*p+*q`, into sequences of simpler operations involving temporary variables, e.g. `t1:=*p; t2:=*q; *r:=t1+t2`. Some IRs even break down this last statement into an add operation followed by a separate store operation.)

5. Define a data flow analysis that can be used to compute two sets of variables for each procedure as a whole: those which are definitely not defined before use, and those which may not be defined before use; these sets can be used 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. Specify the flow functions for all the kinds of assignment statements of the IR from the previous question. Explain how to use the information computed by your analysis at each program point to construct the two sets specified above for output to the user.

6. Define a dataflow analysis to compute, at each program point, the set of (local) variables that may have had their address taken previously. Use a lattice-theoretic formalism. Assume the IR given above, augmented with the following additional kind of right-hand-side:

```
RHS   ::= ... | "&" Var     address of a variable
```

   Explain how bit-vectors could be used to implement this analysis efficiently. Define how the results of this analysis can be used to define `may-point-to(p)`, the set of variables that variable `p` may point to.