CS243 Assignment 3: More Dataflow Analysis

Due: February 4th, Thursday, 3:00 pm

This homework contains a combination of written assignment and gradiance quizzes. For the gradiance quizzes please refer to http://www.gradiance.com. For the written assignment, every student must hand in his or her homework. Please submit your homework in-class. SCPD students may submit their homework by e-mail via scp-distribution@lists.stanford.edu or give your homework to the courier.

Q1. Apply lazy code motion to the following program. You do not need to show the intermediate steps, just show the optimized code. You may add basic blocks to the flow graph, but only show those that are not empty in your solution.
Q2. You are given the task of optimizing the code given below. You are only allowed to run the following three optimization techniques:
- PRE (as discussed in class)
- Constant Propagation (as discussed in class)
- Copy Propagation (as discussed in Section 9.1.5 of the textbook)
in any order and multiple time if necessary.

a. What is the order in which you executed them to produce the best optimized code by running a minimum number of analyses?
b. What is the final optimized program?
Q3. Consider the following control flow graph:

Part 1
a. Apply the Partial Redundant Expression (PRE) elimination on the above graph
b. If the loop runs for ‘N’ iterations, compute the number of additions and assignments in the above program

Part 2
The Partial Redundant Assignment Elimination (PRAE) is a technique that is slightly more powerful than PRE in that it also takes into account the lhs of assignments. An assignment instance a<sub>i</sub> of assignment pattern a (‘x := e’) is considered redundant if all paths from Start to a<sub>i</sub> has seen an instance of ‘a’, a<sub>j</sub>, and there are no statements between a<sub>i</sub> and a<sub>j</sub> which re-define the variable x or operand of the expression e. Partial Redundancy is taken into account when not all paths from Start have seen ‘a’ and therefore, additional assignments are added to make the assignment completely redundant.

a. How would you formulate the PRAE algorithm? You don’t need to describe the algorithm, just apply it to the above graph and show the result.
b. If the loop runs for ‘N’ iterations, compute the number of additions and assignments in the above program. Compare the result with the answer to Part 1-(b).

Part 3
a. Does it make sense to combine both the techniques as a single technique? (Yes/No)
b. If you answer “yes” to part (a), show the resultant graph of applying the combined technique. Compute the number of additions and assignments and compare the results to Parts 1&2-(b)
c. If you answer “no” to part (a), provide a counter-argument against combining them
Q4. Consider the following control flow graph:

a. Draw the dominator tree for this graph.
b. Find the back-edges and natural loops in this graph.

The Dominance Frontier DF(d) of a node d is defined as the set of all nodes n such that d dominates an immediate predecessor of n, but d does not strictly dominate n. It is the boundary of the flow graph where the dominance of d stops.

We define DOM_BY(n) as the set of all nodes that is dominated by n and SUCC(n) as the list of nodes that have an edge from node n.

c. Represent DF(n) in terms of DOM_BY(n) and SUCC(n).
d. Using the relation to your answer in (c) compute the dominance frontier for all the nodes in the above graph.
Q5. Dominance frontier is a very useful relation to compute the Static Single Assignment (SSA) form of a program. In the static single assignment form, every use of a variable can be traced back to a single definition in the program and therefore is very powerful in doing several analysis like reaching definitions, liveness analysis, etc. in linear time.

Static single assignment resolves the multiple def-use of a single variable by numbering each definition of the variable and introducing the concept of $\Phi$-functions to resolve control flow paths. An example of the static single assignment form is given below:

In the example above, the $\Phi$-function generates a new definition for $X$, $X_2$ which chooses $X_0$ or $X_1$ based on the control path it arrived from. Given a control flow graph, the SSA form can be trivially generated by numbering all the definitions in the program and generating $\Phi$-functions at all meet points in the program. However, that creates unnecessary $\Phi$ definitions that are only used to only generate new $\Phi$ definitions.

To generate the minimal SSA form, the $\Phi$-functions should be inserted at the dominance frontier of node where the variable is defined. The algorithm recursively places $\Phi$-functions at the dominance frontier of the definitions of the variables till no more $\Phi$-functions are added to the original program.
We provide below an example flow graph where SSA has been applied.

```
x=1
  c=x
    d=x
      x==10?
        print x
  a=x
    b=x
    e=x
      x++
    x=2
```

For the given flow graph, the non-minimal or naive SSA which inserts $\Phi$ functions at all meet points is shown below:

```
x_{i}=1
  c=x_{i}
    x_{4} = \Phi(x_{i})
  a=x_{i}
    x_{5} = \Phi(x_{i}, x_{2})
      b=x_{5}
        x_{6} = \Phi(x_{4}, x_{5}, x_{3})
          d=x_{6}
            x_{7} = \Phi(x_{6}, x_{8})
              x_{8} = \Phi(x_{6}, x_{9})
                x_{9} = \Phi(x_{6}, x_{10})
                  x_{10} = \Phi(x_{6}, x_{11})
                    x_{11} = 10?
                      print x_{r}
            x_{7} = x_{7} + 1
```

```
x_{j}=3
```

```
However, the minimal SSA form for the program is shown below:

a. Apply the algorithm described above and generate the SSA form for the given control flow graph:

b. The SSA makes it easy to find constants and propagate them. Show the result of constant propagation for the above graph by taking advantage of the SSA form.