CS243 Homework 3

Winter 2019

Due: February 6, 2019 at 4:30 pm

Directions:

• Submit via Gradescope.
• You may use up to two of your remaining late days for this assignment, for a late deadline of February 8, 2019 at 4:30 pm.
• Remember to complete the corresponding Gradiance quizzes by the start of class on the due date. There are no late days for Gradiance.
• This is an individual assignment. You are allowed to discuss the homework with others, but you must write the solution individually. If you look up any material in the textbook or online, you should cite it appropriately.
Problem 1. Apply PRE to the following program. Assume that variables $w$, $v$, $x$, and $r$ are used in other portions of the code (not shown), possibly in the same basic block. 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 (existing basic blocks are not empty, even if they appear to be).
**Problem 2.** Consider the following flow graph

1. Draw the dominator tree for the graph.

2. What are the back edges and natural loops of the graph?

3. A node $d$ strictly dominates a node $n$ if $d$ dominates $n$ and $d \neq n$. The dominance frontier of a basic block $b$, $DF(b)$, is the set of all blocks $n$ such that (1) $b$ dominates an immediate predecessor of $n$ and (2) $b$ does not strictly dominate $n$. This is the boundary of the flow graph wherein the dominance of $b$ terminates.

   (a) Let $DOM\_BY(b)$ be the set of all basic blocks dominated by a basic block $b$ and $SUCC(b)$ be the set of blocks $s$ such that there exists an edge $b \rightarrow s$. Express $DF(b)$ in terms of $DOM\_BY$ and $SUCC$.

   (b) Can a block be in its own dominance frontier? If not, provide a brief explanation. Otherwise, provide an example of such a block in the graph.
Problem 3. The dominance frontier introduced in problem 2 can be helpful when computing the *Static Single Assignment (SSA)* form of a program. To resolve the multiple def-use of a single variable, static single assignment numbers each definition of the variable and uses a $\phi$ function to merge multiple definitions along multiple control paths into a single definition. Every assignment can then be traced back to a single definition in the program, making it a powerful tool in performing analyses such as liveness and reaching definitions efficiently. Below are some examples of programs and their SSA forms.

Given a control flow graph, the SSA form can be generated by numbering all definitions and creating $\phi$ functions at all meet points in the graph, then numbering variables used on the RHS by their most recent definition. This, however, can create unnecessary $\phi$ definitions that are only used to generate a new $\phi$ definition down the line. Consider the following example, which contains an extraneous definition $x_3$:
In minimal SSA form, we could do away with $x_3 = \phi(x_1)$ as follows:

Intuitively, the dominance frontier indicates a meet point of the control flow in which multiple definitions can come together. We can thus generate the minimal SSA form by instead inserting $\phi$ functions at the dominance frontier of nodes at which the variable is defined. In this algorithm, $\phi$ functions are recursively placed at the dominance frontier of the definitions of the variables until no more $\phi$ functions are added to the program.

For the following program,

1. Use dominance frontiers and the algorithm described to generate its minimal SSA form.

2. Show the results of constant propagation over the resulting SSA form.
**Problem 4.** For the following control flow graph, perform register allocation. Show the results of the following steps.

1. Assign each definition and use of a variable to a live range. For example, all instances of $A$ must be replaced with either $A_1$ or $A_2$ to signify one of two live ranges.

2. Draw the register interference graph with lines between nodes that represent live ranges.

3. Apply the heuristic-based register allocation algorithm with for a machine with 3 registers. Show the resulting “stack” of registers and show which ones, if any, are marked as spilled.

4. Assign the live ranges to registers.

```plaintext
A = 5;
B = 3;
C = 2;

D = A + 2;
A = 9;

E = 2 * B;
E = 2 * C;

D = D + A;
E = E + A;
return D + E;
```
Problem 5. Observe the control-flow graph below and answer the following questions.

1. What is the largest number of overlapping live ranges seen at any program point?

2. What is the minimum number of registers you need in order to successfully assign all variables without spilling?

3. Now, imagine that, as part of register allocation, you can insert `MOVE x y` operations that copy a value from register x to another register y. Can you allocate all of the variables with fewer registers than before?
Problem 6. You are given the task of optimizing the code given below. You are only allowed to run the following four optimization techniques in any order and multiple times if necessary:

- PRE (as discussed in class)
- Constant Propagation (as discussed in class)
- Copy Propagation (as discussed in Section 9.1.5 of the textbook)
- Dead Code Elimination (liveness analysis, as discussed in class and in Homework 2)

You cannot modify the control flow graph or eliminate empty basic blocks, except to preprocess it for PRE. As in joeq, assume that an expression can take both registers and constants.

1. What is the order in which you should execute them to produce the best optimized code by running a minimum number of analyses?

2. What is the final optimized program?