CS243 Homework 1

Winter 2017

Due: January 24, 2017

Directions:

- Complete the following problems and hand them in at the beginning of class with your name and SUID at the top.
- Remember to complete the corresponding Gradiance quizzes by the start of class on the due date. **There are no late days for Gradiance.**
- If you need to use one late day, hand in your assignment by 3pm Wednesday in Gates 441. Slide it under the door if it is locked. If you need two days, hand it in at the beginning of class Thursday.

Problem 1. Indicate which of the following are meet operators. If not, justify your answer by listing the properties that fail to hold.

1. Component-wise minimum for tuples of integers
2. Addition over complex numbers
3. Multiplication over complex numbers
4. Arithmetic mean over real numbers
5. AND over booleans

Problem 2. True or False? Briefly justify your answer in 1 to 5 lines.

1. The MFP solution for all monotone dataflow problems is the MOP solution.
2. There is at most one fixed point solution for a dataflow problem.
3. A semi-lattice can have more than one bottom element.
4. A semi-lattice must have a top element.

Problem 3. Initial Values in Dataflow Analysis

This question asks you to think about how changes to initial values in a data flow analysis can affect the result. Recall that an answer to a data flow problem is considered "safe" if it is no bigger than the ideal solution.

Suppose you have defined a forward dataflow algorithm that is distributive and has finite descending chains. You accidentally initialized OUT[B] to $\perp$ for all nodes other than the ENTRY node.

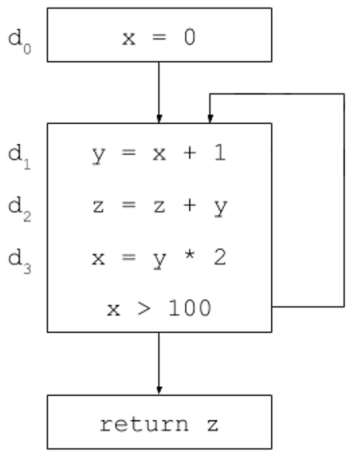
1. Will your algorithm give a safe answer for all flow graphs?
2. If not, will it give a safe answer for some flow graphs? If it will, give an example.
3. Will your algorithm give the MOP solution for all flow graphs?
4. If not, will it give the MOP solution for some flow graphs? If it will, give an example.

Problem 4. Live Range Analysis

We say that a path is *definition free* with respect to a variable y if there is no definition of variable y along that path. The live range of a definition $d : y = x + z$ that defines variable y consists of all the program points p such that both of the following hold:

1. There is a path from d to p that is definition free with respect to y.
2. There is a path from p to q that is definition free with respect to y, where q is a statement that uses the variable y.

Intuitively, the live range of a definition includes points along all subsequent paths until either it is overwritten by a new definition or the variable is no longer used along that path. The concept of live ranges can be used in register allocation: if live ranges of two definitions do not intersect, they can be assigned to the same register. Consider the following program:

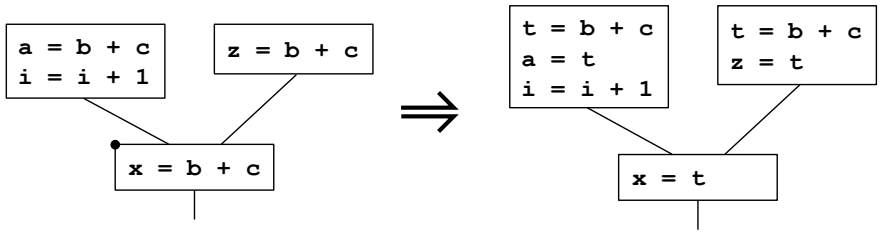


In this example, definitions d_0 and d_3 define the variable x, d_1 defines y and d_2 defines z. The live ranges of d_0, d_1, and d_3 do not intersect and can use the same register. The entire program can run with just 2 registers!

Describe an analysis that computes the live range for each definition in a program. You may use algorithms discussed in class, in which case, you don't have to describe those algorithms again, but be specific on how you compute live range.

Problem 5. Available Expression Analysis

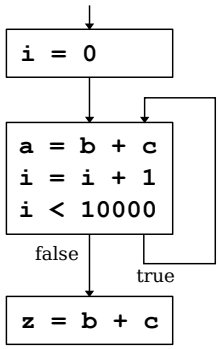
1. An *available expression* at a point p is one which has been computed on all paths up to that point, without a subsequent overwrite of one of the operands of the expression. This can be used to eliminate some repeated computations by writing the computed value to a common temporary variable. For example, using the fact that `b + c` is an available expression at the indicated program point on the left, we can rewrite to the equivalent program on the right:



   Note that if we change `i = i + 1` to `b = i + 1` in the left block, the expression is no longer available at the marked point because one of the operands has been overwritten on that path.

   Describe a dataflow analysis that computes the available expressions at each program point.

2. Consider the following program:



   (a) How many times does the program compute `b + c`?

   (b) Assume you run your available expressions analysis on the program and use the results to rewrite the program as in the previous example. How many times does the program now compute `b + c`?