CS 243 Homework 1

[YOUR NAME] - [YOUR SUNET ID]@stanford.edu

Due: January 24, 2024, 11:59 pm

Directions:

- Before starting this assignment, read textbook section 9.2.6 (pp. 610–614) to learn about the Available Expressions analysis. This will be needed for both Problem 3 and the Gradiance quizzes.
- Submit written answers via Gradescope.
- Complete the corresponding Gradiance quizzes by the due date.
- You have **two late days** on assignments for the entire quarter. See course website for details. **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. Indicate which of the following operator–domain pairs defines a semilattice. If so, also define the **top** ($\top$) and **bottom** ($\bot$) elements of the associated semilattice, if they exist. If not, justify your answer by listing at least one property of semilattice that fails to hold.

1. Set intersection over {{1}, {2}, {1, 2}}
2. Set union over the power set of $\mathbb{Z}^+$ (all positive integers)
3. Boolean implication ($\implies$) operator over boolean values $\{T, F\}$
4. The GCD (Greatest Common Divisor) function over $\{2, 4, 8\}$
5. Geometric mean ($a \wedge b = \sqrt{a \cdot b}$) over $\mathbb{R}^+$ (all positive real numbers)
6. Longest common prefix over S_5: the set of all strings of length = 5.
7. The meet operator $\wedge$, defined on $\{0, 1, 2, 3\}$, is given by the following table (the row and column headers specify the first and second operands, respectively):

$\wedge$	**0**	**1**	**2**	**3**
0	0	0	3	3
1	0	1	2	3
2	3	2	2	3
3	3	3	3	3

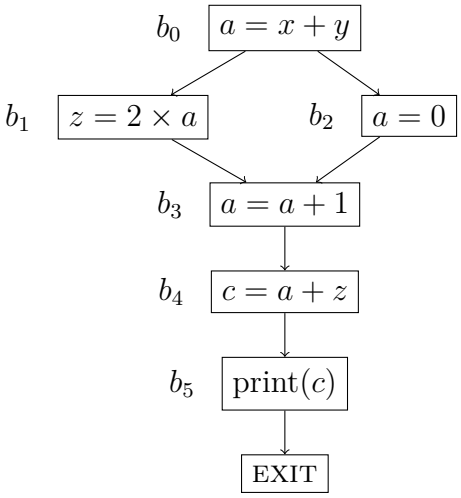
Problem 2. Live Range Analysis.

A path is *definition-free* with respect to a variable y if there does not exist a definition of variable y along that path. The live range of a definition d: $y = x + z$ that defines variable y includes all the program points p such that:

1. There is a path from d to p that is definition-free with respect to y, and
2. There is a path from p to q, a statement that uses (i.e., reads) the variable y, that is definition-free with respect to y.

To clarify, for any statement d within basic block b, there is no path from d to entry(b) (unless b participates in a cycle in the CFG), but there is a path from d to exit(b).

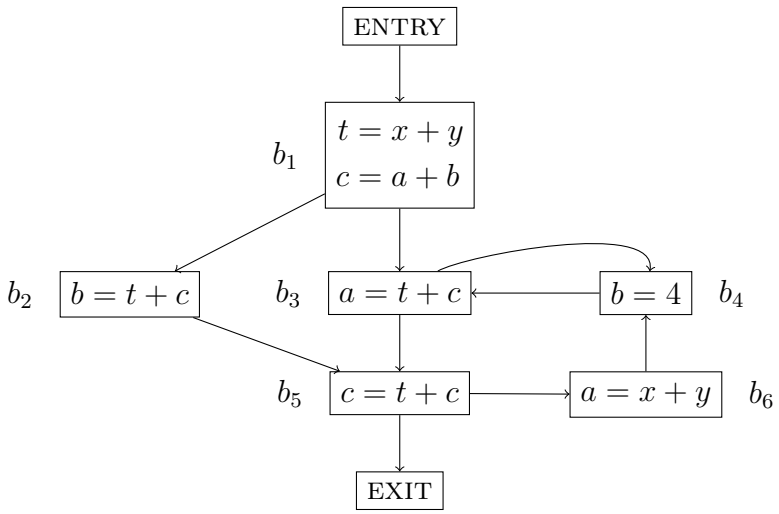
Intuitively, the live range of a definition consists of points along all subsequent paths until the variable defined is guaranteed never to be used before redefinition (or exit) along all paths. This concept is applicable to register allocation: two definitions can be assigned to the same register if their live ranges do not intersect.



In the above example, the live range of definition b_0 is exit(b_0), entry(b_1), exit(b_1), and entry(b_3). Notably, entry(b_2) is *not* part of the live range of definition b_0 (think of why this is the case). Similarly, the live range of the definition b_4 is exit(b_4), entry(b_5). The two live ranges do not intersect, so c can reuse a's register.

Briefly describe an analysis that computes the live range for each definition in a program. *Hint: you may use algorithms discussed in class.*

Problem 3. Read textbook section 9.2.6 (pp. 610–614) to learn about the Available Expressions analysis. Compute the available expressions on entry and exit for each basic block in the following flow graph.



Problem 4. True or False? Briefly justify your answer.

1. As discussed in lecture 3, $\text{MOP}(b) = \wedge f_{p_i}$ for all paths p_i reaching block b. Consider a program where the maximum path length among any of these paths p_i is N. You are applying the iterative data flow algorithm discussed in class to a monotone data flow framework. Let $\text{IN}_N[b]$ be the value of $\text{IN}[b]$ after N iterations of the algorithm, $\text{IN}_N[b] \leq \text{MOP}(b)$ holds true for all blocks b.

2. Let S be the set $\{a, b, c, d, e\}$. Consider a semi-lattice where the partial order is defined by the subset relation $\subseteq$ over the power set of S, $\mathcal{P}(S)$. Let $f : \mathcal{P}(S) \to \mathcal{P}(S)$ be the complement function, that is, $f(A) = A^c$. The data flow framework with f and the partial order $\subseteq$ is a distributive framework.

3. A distributive data flow framework is also a monotone data flow framework.

4. If the semilattice of a data flow framework has an infinite domain, then the iterative algorithm cannot converge to a fixed point solution.

5. Suppose $f : S \to S$ and $g : S \to S$ are monotone functions with respect to some partial order $\leq$. Then $f \circ g$ is also monotone (where $(f \circ g)(x) = f(g(x))$).

6. Suppose we have a partial order defined by the superset ($\supseteq$) relation over the power set of $\mathbb{Z}$, $\mathcal{P}(\mathbb{Z})$. We define function $f : \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$ as $f(S) = ((S \cap \{1, 2, 3, 4, 5\}) \cup \{6\})$. f is a monotone function with respect to this partial order. *Hint: Use your answer from the previous part.*

7. The Live Variable analysis discussed in class can be applied to functions that use both local and global variables without needing any modifications.

Problem 5. Initial Values in Data flow 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 greater than the ideal solution.

Suppose you have defined a forward data flow algorithm that has a monotone framework and finite descending chains. You accidentally initialized OUT[B] to $\perp$ for all nodes other than ENTRY.

1. Will your algorithm still give a safe answer for all flow graphs? If so, please explain. If not, provide a counterexample.

2. Will your algorithm give the MOP solution for all flow graphs? If so, please explain. If not, provide a counterexample.

3. If not, will it give the MOP solution for some flow graphs? If it will, provide an example.

Problem 6. Detecting memory leaks in a toy instruction set

A common challenge associated with the use of pointers in code is the occurrence of memory leaks. When a pointer points to a particular memory location, any subsequent reassignment of the pointer to a different location or reallocation of the pointer may result in the abandonment of the old memory location without freeing it, leading to a memory leak. This concern becomes particularly significant in long-running programs as leaked data can accumulate and cause them to fail by running out of memory. Hence, it becomes imperative to proactively address potential memory leaks within a program.

For this problem, let us consider a toy instruction set where only local, scalar variables can be pointers. In other words, you cannot store pointers into allocated data, there are no complex data structures like linked lists and trees. The available instructions are:

1. `p = allocate()` : Allocate memory and hold the reference in pointer `p`.
2. `free(p)` : Free the memory pointed by pointer `p`.
3. `move(p, q)` : Transfer the memory location referred by `p` to `q`, so that now `q` holds the reference to the memory location and `p` stores null. In C/C++ syntax, this is essentially equivalent to: `q = p; p = null;`
4. No other instructions can allocate memory, free memory, change the values of pointer variables, or assign pointers to any variables.

In simple terms, a memory location will be pointed to by **at most one pointer** at a time.

Your task is to warn programmers of any instructions in the program that MAY cause a memory leak. Specifically, you need to flag any instruction that can potentially cause a memory location to never be freed.

(1) Define a data flow analysis to solve this problem by filling in the table below, and (2) Specify how you would use the data flow results to issue warnings on specific vulnerable statements. You may treat each instruction as a basic block.

There are other potential problems such as dereferencing an uninitialized pointer, use-after-free, etc. You may ignore such errors in this analysis.

Direction of your analysis (forward/backward)	
Lattice elements and meaning	
Is there a top element? If yes, what is it?	
Is there a bottom element? If yes, what is it?	
Meet operator	
Transfer function	
Boundary condition	
Interior points	

Issue warning for basic block b if: