CS243 Homework 1

Winter 2021

Due: January 27, 2021 at 4:00 pm

Directions:

- This homework includes a Gradiance quiz (please see the website) and the following questions, to be submitted via Gradescope.
- You may use up to two of your remaining late days for this assignment, for a late deadline of January 29, 2021 at 4:00 pm. However, you need to complete the Gradiance quiz 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. Indicate which of the following operators defines a semi-lattice. Define the top and bottom element of the associated semi-lattice, if they exist. If not, justify your answer by listing the properties that fail to hold.

1. Set Union.
2. Set symmetric difference (i.e. $(A \setminus B) \cup (B \setminus A)$).
3. Component-wise addition of tuples of natural numbers.
4. Or over boolean values ($\{T, F\}$).
5. Mediant ($\frac{a}{b} \oplus \frac{c}{d} = \frac{a+c}{b+d}$) over positive rational numbers (including 0) where a, b, c, d are non-negative integers. *Hint: Be sure to first check that this operation is well-defined and closed over the positive rational numbers!*
6. Arithmetic mean over real numbers (for $a, b \in \mathbb{R}$, define $a \wedge b = \frac{a+b}{2}$).
7. The GCD (Greatest Common Divisor) function on non-negative, non-zero integers.
8. Cross product on three-dimensional ($\mathbb{R}^3$) vectors, defined as follows:

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

Problem 2. True or False? Briefly justify your answer.

1. A monotone framework is also a distributive framework.
2. If the semi-lattice of a data-flow framework has a finite domain, then the iterative algorithm must converge to some fixed point solution.
3. A semi-lattice can have multiple top elements.
4. Suppose $f : S \to S$ and $g : S \to S$ are monotonic functions with respect to some partial order $\leq$. Then $f \circ g$ is also monotonic.

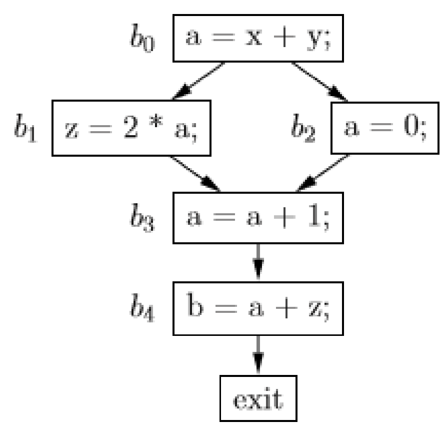
   Hint: a function $f : X \to Y$, f is monotonic if and only if $\forall a, b \in X, a \leq b$ implies $f(a) \leq f(b)$.
5. Suppose we have a partial-order defined by the subset ($\subseteq$) relation over all sets of integers, $\mathcal{P}(\mathbb{Z})$. We define function $f : \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$ as $f(S) = (S \cup \{1\}) \setminus \{2\}$. f is a monotonic function with respect to this partial order.

   Hint: Use your answer from the previous part.

Problem 3. 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 the variable y, that is definition free with respect to y.

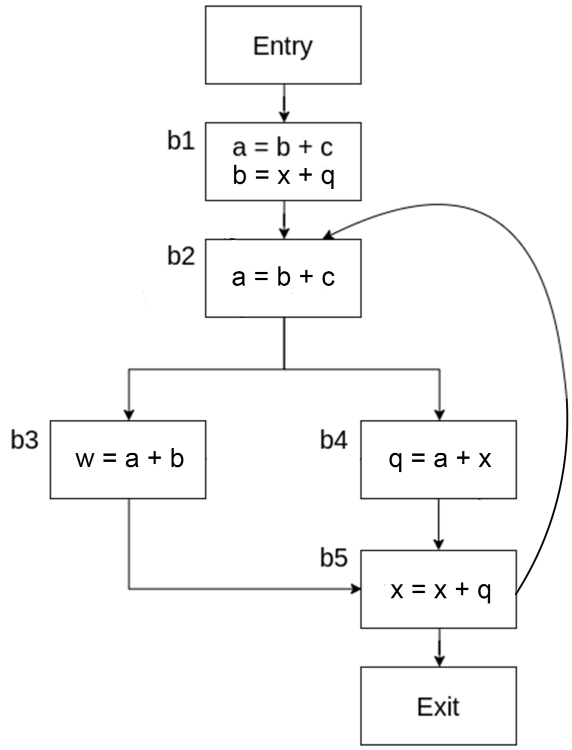
Intuitively, the live range of a definition consists of points along all subsequent paths until either the variable is no longer used along that path or a new definition overwrites it. This concept is applicable to register allocation: two definitions can be assigned to the same register if their live ranges do not intersect.



In the above example, the live range of definition $a = x + y$ is exit(b_0), entry(b_1), exit(b_1) and entry(b_3). Similarly, the live range of the definition $b = a + z$ is empty. The two live ranges do not intersect, so b can reuse a's register.

Describe an analysis that computes the live range for each definition in a program. You may use algorithms discussed in class.

Problem 4. Compute the available expressions (Chapter 9.2.6 in ALSU) on entry and exit for each basic block in the following flow graph:



Problem 5. Understanding 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 bigger than the ideal solution.

Suppose you have defined a forward data flow algorithm that is monotone and has 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 MFP solution for all flow graphs? If so, please explain. If not, provide a counterexample.

3. If your answer to 2 is no, will it give the MFP solution for some flow graphs? If it will, provide an example.

Problem 6. Detecting Errors in Interrupt Handlers

Part A. Warn on Un-elevated Uses of Privileged Instructions

Interrupt handlers are blocks of code that are run in response to events detected by a processor that need software attention. These handlers are run in a special "interrupt" execution context with elevated access to hardware, allowing them to execute privileged instructions. Consider a simplified language with the following constructs:

1. `begin_interrupt`: this enters an interrupt context.

2. `end_interrupt`: this exits an interrupt context.

3. `read_processor_flags`: this is an privileged instruction. It must be run from inside an interrupt execution context.

4. all other instructions that are not privileged instructions and do not enter or exit interrupt contexts.

You may treat each instruction as a basic block.

Your task is to warn programmers about any potential use of the privileged instruction `read_processor_flags` outside of an interrupt context. (1) Define a data flow analysis to solve this problem by filling out the table below, and (2) Specify how you use the data flow results to issue a warning on each potentially invalid use of `read_processor_flags`. Since invalid uses of this instruction can completely halt your processor, you want to warn whenever there is a potential path that could result in un-elevated uses of `read_processor_flags`.

There are other errors like nested entry of interrupt contexts (which may be erroneous depending on the hardware), or calling `end_interrupt` without a corresponding `begin_interrupt`, but you may ignore such errors for this analysis.

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

Part B. Detecting Nested Interrupt Contexts

Now you would like to warn programmers about nested entry into interrupt contexts as this is not supported by your processor. In other words, warnings are given if there are any potential multiple `begin_interrupt`'s in a row, without intervening `end_interrupts`. (1) Define a data flow analysis to solve this problem by filling out the table below, and (2) Specify how you use the data flow results to issue a warning on each potentially extraneous `begin_interrupt` command. Since this error can also cause your processor to halt, you want to warn on any *potential* spurious `begin_interrupt`'s.

You may treat each instruction as a basic block.

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