CS243 Homework 3

Winter 2020

Due: February 5th, 2020 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 7th, 2020 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. 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.

   See graph below.

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

   See graph below.

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.

   There are many valid answers, one possible ordering: E2 D2 B1 C1 A1 D1 E1.

4. Assign the live ranges to registers.

   There are many valid answers, one possible assignment:
   \((E1, R0), (D1, R1), (A1, R2), (C1, R0), (B1, R0), (D2, R1), (E2, R0)\)

---

[Diagram of control flow graph]
Live ranges graph:

```
Entry
    \[ A_1 = 10; \]
    \[ B_1 = 7; \]
    \[ C_1 = 4; \]
    \[ C_1 = 3; \]
    \[ D_1 = A_1 + 4; \]
    \[ A_1 = 2; \]
    \[ B_1 = 2; \]
    \[ E_1 = C_1 + 2; \]
    \[ E_1 = B_1 + 1; \]
    \[ D_2 = D_1 + A_1; \]
    \[ E_2 = E_1 + A_1; \]
    \[ \text{return} \ D_2 + E_2; \]
Exit
```

Interference graph:
Problem 2. 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?

   Two live ranges.

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

   Three registers, due to the interference graph odd-length cycle between the live ranges of A, B, and C.

3. Now, imagine that, as part of register allocation, you can insert \texttt{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? If so, how many would it require?

   Two registers. For example, you can insert a \texttt{MOVE} after \ldots \, = \, B in the middle of the second basic block to swap A’s register, which allows you to reuse the now free register for the allocation of C, breaking the cycle.
Problem 3. Apply PRE to the following program. Assume that variables $s$, $r$, $x$, $y$ and $z$ 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).
Solution:

Entry

- $t = a + b$

- $x = t$

- $s = t$

- $y = t$

- $a = \text{read}()$

- $b = \text{read}()$

- $r = t$

Exit
Problem 4. Consider the following flow graph.

1. Draw the dominator tree for the graph.

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

\[ B3 \rightarrow B2: \{B2, B3\} \]

\[ B8 \rightarrow B1: \{B1, B2, B3, B4, B5, B6, B7, B8, B9\} \]

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

\[ DF(n) = \bigcup_{d \in \text{DOM\_BY}(n)} \text{SUCC}(d) - (\text{DOM\_BY}(n) - \{n\}) \]

(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. Yes, for instance, \( B2 \) is in its own dominance frontier, as \( B2 \) dominates \( B3 \), a predecessor of \( B2 \) and, is not a strict dominator of itself.
Problem 5. Your task is to optimize the code 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 pre-process it for PRE. As in JoeQ, assume that expressions can take both registers and constants.

1. What are the optimizations and their order to produce the best optimized code for this specific program? Remember that you may run the same optimization more than once.

2. What is the final optimized program?

Constant propagation and dead code elimination will get rid of almost all code. You could end up doing a lot of passes with a different ordering.
entry

q = read()

q = q + 1

m = 3 + q

b = 3 + m

Return b
Problem 6. Warnings on Possible Exceptions.

Your task is to create warnings on square root operations that may be applied to negative numbers, and array accesses that may be using a negative index. To keep the problem simple, this language only allows assignments to arrays and not reads from arrays. Basic blocks in your program can contain one of the following operations, where $x$ and $y$ are integer variables, and $A$ is an array variable:

- $x = c$, where $c$ is an integer constant.
- $x = -y$, which negates the value of $y$.
- $x = \text{sqr}(y)$, which squares $y$.
- $x = \sqrt{y}$, which takes the square root of $y$.
- $A[y] = x$, which assigns $x$ to the $y$th element in array $A$; $y$ cannot be negative.

For every square root operation and array access, we want to show a warning if it could potentially cause an exception due to the input being negative. Describe an analysis that can surface such warnings. First describe your data-flow analysis. Then, describe how the data-flow results can be used to show warnings to the programmer.

When describing your data-flow analysis, fill in the following table.
| Direction of your analysis (forward/backward) | Forwards. |
| Lattice elements and meaning | \{\} - A variable of undefined value. \{0\} - A variable is exactly zero. \{+\} - A variable is positive. \{−\} - A variable is negative. \{0, +\} - A variable is non-negative. \{−, +\} - A variable is non-zero. \{−, 0, +\} - A variable can be any value. |
| 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? | \{−, 0, +\} |
| Transfer function of a basic block | 
| \(x = c;\) | Set \(x\) to \{0\} if \(c = 0\), \{−\} if \(c < 0\), or \{+\} if \(c > 0\) |
| \(x = −y;\) | We map the lattice element corresponding to \(y\) to an lattice element corresponding to \(x\). \{−\} → \{+\}, \{+\} → \{−\}, \{−, 0\} → \{0, +\}, \{0, +\} → \{−, 0\}. The other values stay the same. |
| \(x = \text{sqr}(y);\) | Define a mapping in a similar manner as above: \{\} → \{0, +\}, \{−\} → \{+\}, \{−, 0\} → \{0, +\}, \{−, +\} → \{+, 0\}, \{−, 0, +\} → \{0, +\}. The other values stay the same. |
| \(x = \sqrt{y};\) | Define a mapping in a similar manner: \{\} → \{−, 0, +\}, \{−\} → \{−, 0, +\}, \{−, 0\} → \{−, 0, +\}, \{−, +\} → \{−, 0, +\}. The other values stay the same. |
| Return | out\[b\] = in\[b\] |
| Boundary condition initialization | set all variables to \{0\} |
| Interior points initialization | set all variables to \{} |
For each basic block containing an array access or square root operation, issue a warning on that statement if $\text{IN}[b][x] = \text{"-"}$, where $x$ is the integer variable operand on the right hand side.