

# CMSC 430 (Spring 2009)

## Practice Problems 5 Solutions

---

### 1. Optimizations

- (a) How can compiler transformations improve a program?

*By reducing program size or number of instructions executed, taking advantage of redundant computations, customizing general purpose code, or undoing high-level abstractions.*

- (b) What does the compiler need to consider when applying optimizations?

*Safety, profitability, and applicability.*

- (c) What are the different scopes of compiler optimizations? What are the tradeoffs when considering what scope of optimizations to use?

*Peephole, local, global, or interprocedural. The basic tradeoff is more complexity and compile time for optimizations with greater scope.*

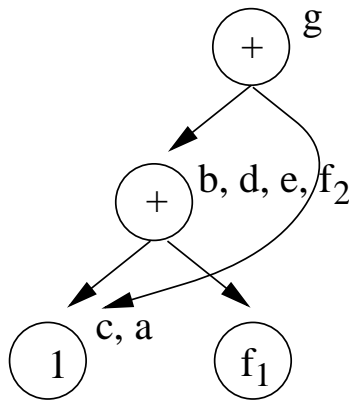
---

### 2. Local optimizations

Consider the following code.

- (1)  $a := 1$
- (2)  $b := f + a$
- (3)  $c := a$
- (4)  $d := f + a$
- (5)  $e := f + c$
- (6)  $f := b$
- (7)  $g := f + a$

- (a) Build a DAG for the code.



*Note that assignment gives a node multiple labels, and may require renaming variables.*

---

### 3. Control flow analysis

For the following problems, consider this code:

```

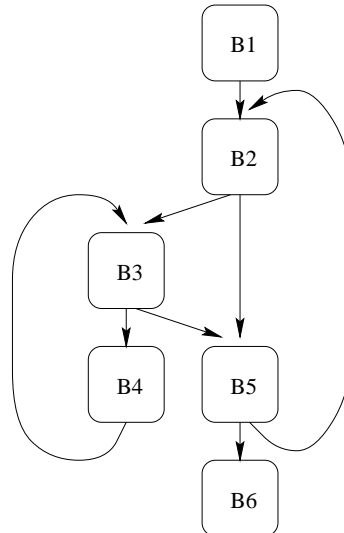
<S1>      a := 1
<S2>      b := 2
<S3>      L1:  c := a + b
<S4>      d := c - a
<S5>      if (...) goto L3
<S6>      L2:  d := b * d
<S7>      if (...) goto L3
<S8>      d := a + b
<S9>      e := e + 1
<S10>     goto L2
<S11>     L3:  b := a + b
<S12>     e := c - a
<S13>     if (...) goto L1
<S14>     a := b * d
<S15>     b := a - d
    
```

- (a) What are the basic blocks?

```

B1 = { S1, S2 }
B2 = { S3, S4, S5 }
B3 = { S6, S7 }
B4 = { S8, S9, S10 }
B5 = { S11, S12, S13 }
B6 = { S14, S15 }
    
```

- (b) What is the control flow graph?



- (c) Depth-first order selects nodes in the order they are visited (start by visiting the root node) and then recursively visiting every child of each node (if the child has not been visited before). Note that the order in which children are visited is random.

What are all the possible results of depth-first traversal on the control flow graph?

$B1, B2, B3, B4, B5, B6$

$B1, B2, B5, B6, B3, B4$

$B1, B2, B3, B5, B6, B4$

- (d) Using depth-first order, is it possible to visit a child before its parent? For which depth-first ordering(s) of the control flow graph does this occur?

$B1, B2, B5, B6, B3, B4$

- (e) Postorder selects nodes (starting from root) *after* visiting every child of that node (if the child has not been visited before). Note that the order in which children are visited is random.

What are all the possible results of Postorder traversal for the control flow graph?

*B6, B5, B4, B3, B2, B1*  
*B4, B6, B5, B3, B2, B1*

- (f) Reverse Postorder simply reverses the node ordering found by a Postorder traversal of the graph. What are the possible Reverse Postorder traversals of the control flow graph?

*B1, B2, B3, B4, B5, B6*  
*B1, B2, B3, B5, B6, B4*

- (g) Using Reverse Postorder, is it possible to visit a child before its parent? Why or why not?

*Yes, because not all parents can be visited first if there are loops (cycles) in the control flow graph.*

*Solving the data flow equations in reverse postorder (B1,B2,B3,B4,B5,B6), we get the following (IN, OUT initially all set to  $\emptyset$ ):*

Block	Data	Pass1	Pass2
B1	IN	$\emptyset$	$\emptyset$
	OUT	a1,b2	a1,b2
B2	IN	a1,b2	a1,b2,c3,d4,d6,b11,e12
	OUT	a1,b2,c3,d4	a1,b2,c3,d4,b11,e12
B3	IN	a1,b2,c3,d4	a1,b2,c3,d4,d8,e9,b11,e12
	OUT	a1,b2,c3,d6	a1,b2,c3,d6,e9,b11,e12
B4	IN	a1,b2,c3,d6	a1,b2,c3,d6,e9,b11,e12
	OUT	a1,b2,c3,d8,e9	a1,b2,c3,d8,e9,b11
B5	IN	a1,b2,c3,d4,d6	a1,b2,c3,d4,d6,e9,b11,e12
	OUT	a1,c3,d4,d6,b11,e12	a1,c3,d4,d6,b11,e12
B6	IN	a1,c3,d4,d6,b11,e12	a1,c3,d4,d6,b11,e12
	OUT	c3,d4,d6,e12,a14,b15	c3,d4,d6,e12,a14,b15

#### 4. Reaching definitions

*Reaching definitions* (RD) for a point in the program  $p$  is defined as the set of definitions of a variable for which there is some path from the definition to  $p$  with no other definition of that variable. Calculate reaching definitions for the code in the control-flow graph problem.

- (a) What is the dataflow equation for RD?

$$RD(b) = \bigcup_{x \in \text{pred}(b)} (GEN(x) \cup (RD(x) - KILL(x)))$$

- (b) In what direction is RD calculated? I.e., does information flow forwards or backwards in the CFG?  
*Forwards.*

- (c) Calculate GEN, KILL for each basic block.

*Note: For reaching definitions, we append to each variable definition its location to distinguish between multiple definitions to the same variable.*

Block	GEN	KILL
B1	a1,b2	a1,b2,b11,a14,b15
B2	c3,d4	c3,d4,d6,d8
B3	d6	d4,d6,d8
B4	d8,e9	d4,d6,d8,e9,e12
B5	b11,e12	b2,e9,b11,e12,b15
B6	a14,b15	a1,b2,b11,a14,b15

- (d) What is a good initial value for RD for each basic block?

*RD for each basic block is initialized to  $\emptyset$  (no reaching definitions).*

- (e) Solve the data-flow equations in reverse Postorder. Show your work.

#### 5. Live variables

*Live variables* (LV) for a point in the program  $p$  is defined as the set of variables  $x$  for which there is some path from  $p$  to a use of  $x$  with no definition to  $x$  on the path. Calculate live variables for the code in the control-flow graph problem.

- (a) We define  $LV(b)$  for a basic block  $b$  to be the set of live variables at the *end* of  $b$ . What is the dataflow equation for LV?

$$LV(b) = \bigcup_{x \in \text{succ}(b)} (GEN(x) \cup (LV(x) - KILL(x)))$$

- (b) In what direction is LV calculated? I.e., does information flow forwards or backwards in the CFG?  
*Backwards.*
- (c) Show GEN, KILL for each basic block.

Block	GEN	KILL
B1	$\emptyset$	a,b
B2	a,b	c,d
B3	b,d	d
B4	a,b,e	d,e
B5	a,b,c	b,e
B6	b,d	a,b

- (d) What is a good initial value for LV for each basic block?

*LV for each basic block is initialized to  $\emptyset$  (no live variables). This is an optimistic assumption for dead code elimination, but also accurately reflects the fact that no variables are live at the end of the program.*

- (e) Solve the data-flow equations in rPostorder. Show your work.

*Solving the data flow equations in postorder*

(B6,B5,B4,B3,B2,B1), we get the following:

Block	Data	Initial	Pass1	Pass2
B6	OUT IN	$\emptyset$ $\emptyset$	$\emptyset$ b,d	$\emptyset$ b,d
B5	OUT IN	$\emptyset$ $\emptyset$	b,d a,b,c,d	a,b,d,e a,b,c,d
B4	OUT IN	$\emptyset$ $\emptyset$	$\emptyset$ or b,d a,b,e	a,b,c,d,e a,b,c,e
B3	OUT IN	$\emptyset$ $\emptyset$	a,b,c,d,e a,b,c,d,e	a,b,c,d,e a,b,c,d,e
B2	OUT IN	$\emptyset$ $\emptyset$	a,b,c,d,e a,b,e	a,b,c,d,e a,b,e
B1	OUT IN	$\emptyset$ $\emptyset$	a,b,e e	a,b,e e

Block	Data	Init	Pass1	Pass2
B1	IN OUT	$\emptyset$ $\emptyset$	$\emptyset$ b+c	$\emptyset$ b+c
B2	IN OUT	$\emptyset$ $\emptyset$	$\emptyset$ b+c	$\emptyset$ b+c
B3	IN OUT	$\emptyset$ $\emptyset$	b+c $\emptyset$	b+c $\emptyset$
B4	IN OUT	$\emptyset$ $\emptyset$	$\emptyset$ $\emptyset$	$\emptyset$ $\emptyset$

In more detail, the solution was calculated as follows (where AVAIL = IN):

Pass1

$$AVAIL(B1) = \emptyset$$

$$\begin{aligned} AVAIL(B2) &= (GEN(B1) \cup (AVAIL(B1) - KILL(B1))) \\ &\quad \cap (GEN(B4) \cup (AVAIL(B4) - KILL(B4))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) \cap (\emptyset \cup (\emptyset - \emptyset)) \\ &= \{b+c\} \cap \emptyset = \emptyset \end{aligned}$$

$$\begin{aligned} AVAIL(B3) &= (GEN(B1) \cup (AVAIL(B1) - KILL(B1))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) = \{b+c\} \end{aligned}$$

$$\begin{aligned} AVAIL(B4) &= (GEN(B2) \cup (AVAIL(B2) - KILL(B2))) \\ &\quad \cap (GEN(B3) \cup (AVAIL(B3) - KILL(B3))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) \cap (\emptyset \cup (\{b+c\} - \{b+c\})) \\ &= \{b+c\} \cap \emptyset = \emptyset \end{aligned}$$

AVAIL(B3) changed, repeat

$$AVAIL(B1) = \emptyset$$

$$\begin{aligned} AVAIL(B2) &= (GEN(B1) \cup (AVAIL(B1) - KILL(B1))) \\ &\quad \cap (GEN(B4) \cup (AVAIL(B4) - KILL(B4))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) \cap (\emptyset \cup (\emptyset - \emptyset)) \\ &= \{b+c\} \cap \emptyset = \emptyset \end{aligned}$$

$$\begin{aligned} AVAIL(B3) &= (GEN(B1) \cup (AVAIL(B1) - KILL(B1))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) = \{b+c\} \end{aligned}$$

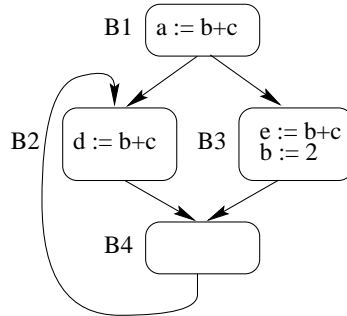
$$\begin{aligned} AVAIL(B4) &= (GEN(B2) \cup (AVAIL(B2) - KILL(B2))) \\ &\quad \cap (GEN(B3) \cup (AVAIL(B3) - KILL(B3))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) \cap (\emptyset \cup (\{b+c\} - \{b+c\})) \\ &= \{b+c\} \cap \emptyset = \emptyset \end{aligned}$$

Pass2 yields no changes to AVAIL, stop

## 6. Available expressions

Available expressions is a data-flow analysis problem whose solution is used to guide global common sub-expression. It calculates AVAIL, the expressions available at the beginning of each basic block.

Consider the following code. Assume that  $b+c$  is the only expression of interest:



(a) What is the data-flow equation for AVAIL?

$$AVAIL(b) = \bigcap_{x \in pred(b)} (GEN(x) \cup (AVAIL(x) - KILL(x)))$$

(b) Give GEN and KILL (needed by AVAIL) for each basic block.

Block	GEN	KILL
B1	$b+c$	$\emptyset$
B2	$b+c$	$\emptyset$
B3	$\emptyset$	$b+c$
B4	$\emptyset$	$\emptyset$

(c) What is a good initial value for AVAIL for each basic block?

Initialize AVAIL to  $\emptyset$  (no expressions available) for all basic blocks,

(d) Calculate AVAIL. Show all steps, including values for AVAIL and the order basic blocks are analyzed. Solving the data flow equations in reverse postorder (B1,B2,B3,B4), we get the following:

## 7. Data-flow lattices

Prove the following properties of lattices:

(a) Show that  $a \leq b$  and  $b \leq c$  implies  $a \leq c$

First, by definition of  $\leq$ , we see  $a \leq b$  gives us  $a \wedge b = a$ .

Similarly, from  $b \leq c$  we get  $b \wedge c = b$ .

Taking  $a \wedge b = a$  and replacing  $b$  with  $(b \wedge c)$ , we get  $a \wedge (b \wedge c) = a$ .

Since  $\wedge$  is associative, we find  $a \wedge (b \wedge c) = (a \wedge b) \wedge c = a \wedge c$ .

But since we also have  $a \wedge (b \wedge c) = a$ , this implies  $a \wedge c = a$ .

By definition of  $\leq$ , this implies  $a \leq c$ .

- (b) Show that  $a \leq (b \wedge c)$  implies  $a \leq b$

First, we show  $(b \wedge c) \leq b$ .

By definition of  $\leq$ , this is true if  $b \wedge (b \wedge c) = b \wedge c$ .

Since  $\wedge$  is associative, we prove it using  $b \wedge (b \wedge c) = (b \wedge b) \wedge c = b \wedge c$ .

Now combining  $a \leq (b \wedge c)$  and  $(b \wedge c) \leq b$  gives us  $a \leq b$ , using the solution to the previous problem.

---

## 8. Data-flow frameworks

- (a) When estimating each of the following sets, tell whether too-large or too-small estimates are conservative. Explain your answer in terms of the intended use of information.

- i. Available expressions

*Too-small estimates are conservative ( $\perp = \emptyset$ ). Common subexpression elimination would not replace some expressions if the estimate is too small, but the answer would be still correct. In comparison, performing CSE when an expression is not available may yield incorrect results.*

- ii. Reaching definitions

*Too-large estimates are conservative ( $\perp = \{ \text{all copy statements} \}$ ). Copy propagation would be restricted if the estimate of copy statements reaching a point is too large, but the result would still be correct. In comparison, estimating too few copy statements reach a given point may enable copy propagation when the right-hand side of a missing copy statement differs from the right-hand side of the copy statements found, yielding incorrect results.*

- iii. Live variables

*Too-large estimates are conservative ( $\perp = \text{all variables}$ ).*

*Dead code elimination uses live variable information to eliminate definitions for variables that are no longer live. Assuming all variables were live would cause dead code elimination to not make any changes to code.*

- (b) What properties are necessary to ensure an iterative data-flow analysis framework terminates?

*The dataflow problem must be monotonic, I.e.,  $f(x \wedge y) \leq f(x) \wedge f(y)$ . All chains (sequences of less-than orderings) in the lattice must also have finite length (true for any lattice with finite height).*

- (c) What properties are necessary to ensure an iterative data-flow analysis framework terminates with the meet-over-all-paths solution?

*The dataflow problem must be distributive. I.e.,  $f(x \wedge y) = f(x) \wedge f(y)$ .*