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.

(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*

(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*

3. Control flow analysis

For the following problems, consider this code:

<B1>
  a := 1
</B1>

<B2>
  b := 2
</B2>

<B3>
  L1: c := a + b
  d := c - a
</B3>

<B4>
  L2: if (...) goto L3
</B4>

<B5>
  L3: b := a + b
  e := c - a
</B5>

<B6>
  L4: if (...) goto L1
</B6>

(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*

(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*

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

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

(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.

\[ \text{GEN} = \bigcup_{x \in \text{pred}(b)} \text{GEN}(x) \]

\[ \text{KILL} = \bigcup_{x \in \text{succ}(b)} \text{KILL}(x) \]

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

\[ RD \text{ for each basic block is initialized to } \emptyset \text{ (no reaching definitions).} \]

(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?

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

(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.

\[ \text{GEN} = \bigcup_{x \in \text{pred}(b)} \text{GEN}(x) \]

\[ \text{KILL} = \bigcup_{x \in \text{succ}(b)} \text{KILL}(x) \]

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

\[ LV \text{ for each basic block is initialized to } \emptyset \text{ (no live variables).} \]

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

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

<table>
<thead>
<tr>
<th>Block</th>
<th>Data</th>
<th>Initial</th>
<th>Pass1</th>
<th>Pass2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B6</td>
<td>OUT</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>IN</td>
<td>0</td>
<td>b,d</td>
<td>b,d</td>
</tr>
<tr>
<td>B5</td>
<td>OUT</td>
<td>0</td>
<td>b,d</td>
<td>a,b,c,d</td>
</tr>
<tr>
<td></td>
<td>IN</td>
<td>0</td>
<td>a,b,c,d,e</td>
<td>a,b,c,d,e</td>
</tr>
<tr>
<td>B4</td>
<td>OUT</td>
<td>0</td>
<td>0 or b,d</td>
<td>a,b,c,d,e</td>
</tr>
<tr>
<td></td>
<td>IN</td>
<td>0</td>
<td>a,b,c,d,e</td>
<td>a,b,c,d,e</td>
</tr>
<tr>
<td>B3</td>
<td>OUT</td>
<td>0</td>
<td>a,b,c,d,e</td>
<td>a,b,c,d,e</td>
</tr>
<tr>
<td></td>
<td>IN</td>
<td>0</td>
<td>a,b,c,d,e</td>
<td>a,b,c,d,e</td>
</tr>
<tr>
<td>B2</td>
<td>OUT</td>
<td>0</td>
<td>a,b,c,d,e</td>
<td>a,b,c,d,e</td>
</tr>
<tr>
<td></td>
<td>IN</td>
<td>0</td>
<td>a,b,c,d,e</td>
<td>a,b,c,d,e</td>
</tr>
<tr>
<td>B1</td>
<td>OUT</td>
<td>0</td>
<td>a,b,c,d,e</td>
<td>a,b,c,d,e</td>
</tr>
<tr>
<td></td>
<td>IN</td>
<td>0</td>
<td>e</td>
<td>e</td>
</tr>
</tbody>
</table>

6. Available expressions

Available expressions is a data-flow analysis problem whose solution is used to guide global common subexpression. 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?

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

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

<table>
<thead>
<tr>
<th>Block</th>
<th>GEN</th>
<th>KILL</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>b+c</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td>b+c</td>
<td>0</td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
<td>b+c</td>
</tr>
<tr>
<td>B4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

Initialize AVAIL to ∅ (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:

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

Pass1

AVAIL(B1) = ∅

AVAIL(B2) = (GEN(B1) ∪ (AVAIL(B1) − KILL(B1))) ∩ (GEN(B4) ∪ (AVAIL(B4) − KILL(B4)))

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

Pass2 yields no changes to AVAIL, stop

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 \( a \land b = a \).

Similarly, from \( b \leq c \) we get \( b \land c = b \).

Taking \( a \land b = a \) and replacing \( b \) with \( (b \land c) \), we get \( a \land (b \land c) = a \).
Since $\land$ is associative, we find $a \land (b \land c) = (a \land b) \land c = a \land c$.

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

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

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

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

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

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

Now combining $a \leq (b \land c)$ and $(b \land c) \leq b$ 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 ($\bot = \emptyset$). 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 ($\bot = \{ \text{all copy statements} \}$).

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 ($\bot = \text{all variables}$).

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., $f(x \land y) \leq f(x) \land f(y)$. 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., $f(x \land y) = f(x) \land f(y)$. 