Lecture 6

More Examples of Data Flow Analysis:
Global Common Subexpression Elimination:
Constant Propagation/Folding

I. Available Expressions Analysis
II. Eliminating CSEs
III. Constant Propagation/Folding

Reading: 9.2.6, 9.4

Global Common Subexpressions

Formulating the Problem

- Domain:
  - a bit vector, with a bit for each textually unique expression in the program
- Forward or Backward?
- Lattice Elements?
- Meet Operator?
  - check commutative, idempotent, associative
- Partial Ordering
- Top?
- Bottom?
- Boundary condition: entry/exit node?
- Initialization for iterative algorithm?

Transfer Functions

- Can use the same equation as reaching definitions
  - out[b] = gen[b] \(\cup\) (in[b] - kill[b])
- Start with the transfer function for a single instruction
  - When does the instruction generate an expression?
  - When does it kill an expression?
- Calculate transfer functions for complete basic blocks
  - Compose individual instruction transfer functions
Composing Transfer Functions

- Derive the transfer function for an entire block
  \[ \text{in1} \]
  \[ \rightarrow \text{out1} = \text{gen1} U (\text{in1} - \text{kill1}) = \text{in2} \]
  \[ \rightarrow \text{out2} = \text{gen2} U (\text{in2} - \text{kill2}) \]

- Since \( \text{out1} = \text{in2} \) we can simplify:
  - \( \text{out2} = \text{gen2} U ((\text{gen1} U (\text{in1} - \text{kill1})) - \text{kill2}) \)
  - \( \text{out2} = \text{gen2} U (\text{gen1} - \text{kill2}) U (\text{in1} - (\text{kill1} U \text{kill2})) \)
  - \( \text{out2} = \text{gen2} U (\text{gen1} - \text{kill2}) U (\text{in1} - (\text{kill2} U (\text{kill1} - \text{gen2}))) \)

- Result
  - gen = \( \text{gen2} U (\text{gen1} - \text{kill2}) \)
  - kill = \( \text{kill2} U (\text{kill1} - \text{gen2}) \)

II. Eliminating CSEs

- Available expressions (across basic blocks)
  - provides the set of expressions available at the start of a block

- Value Numbering (within basic block)
  - Initialize Values table with available expressions

- If CSE is an “available expression”, then transform the code
  - Original destination may be:
    - a temporary register
    - overwritten
    - different from the variables on other paths
  - One solution: Copy the expression to a new variable at each evaluation reaching the redundant use

III. Limitation: Textually Identical Expressions

- Commutative operations
  \[ \text{add t1} = x, y \]
  \[ \text{add t2} = y, x \]
  \[ \rightarrow \text{sort the operands} \]

- Examples
  - Expressions with more than two operands
    \[ \text{add t1} = x, y \]
    \[ \text{add t2} = t1, z \]
    \[ \text{add t3} = y, x \]
    \[ \text{add t4} = t3, z \]
    \[ \text{add t5} = x, y \]
    \[ \text{add t6} = t5, z \]
  - Textually different expressions may be equivalent
    \[ \text{add t1} = x, y \]
    \[ \text{beq t1, t2, t1} \]
    \[ \text{cpy z} = x \]
    \[ \text{add t3} = z, y \]
Another Example

\[
x = 1 \\
y = 1 \\
x = x + 1 \\
y = y + 1
\]

III. Constant Propagation/Folding

- At every basic block boundary, for each variable \( v \)
  - determine if \( v \) is a constant
  - if so, what is the value?

\[
e = 1 \\
x = 2 \\
x = x + e \\
e = 3 \\
p = e + 4
\]
Equivalent Definition

- Meet Operation:

<table>
<thead>
<tr>
<th>v1</th>
<th>v2</th>
<th>v1 ( \land ) v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>undef</td>
<td>undef</td>
<td>undef</td>
</tr>
<tr>
<td>c1</td>
<td>undef</td>
<td>c1</td>
</tr>
<tr>
<td>NAC</td>
<td>c2</td>
<td>NAC</td>
</tr>
<tr>
<td>NAC</td>
<td>undef</td>
<td>NAC</td>
</tr>
</tbody>
</table>

- Note: \( \text{undef} \land c2 = c2! \)

Example

\[
x = 2
\]

Transfer Function

- Assume a basic block has only 1 instruction
- Let \( \text{IN}[b,x], \text{OUT}[b,x] \)
  - be the information for variable \( x \) at entry and exit of basic block \( b \)
- \( \text{OUT}[\text{entry}, x] = \text{undef}, \text{for all } x \).
- Non-assignment instructions: \( \text{OUT}[b,x] = \text{IN}[b,x] \)
- Assignment instructions: (next page)

Constant Propagation (Cont.)

- Let an assignment be of the form \( x_3 = x_1 + x_2 \)
  - \( + \) represents a generic operator
  - \( \text{OUT}[b,x] = \text{IN}[b,x], \text{if } x = x_1 \)

<table>
<thead>
<tr>
<th>( \text{IN}[b,x_i] )</th>
<th>( \text{IN}[b,x_j] )</th>
<th>( \text{OUT}[b,x_k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>undef</td>
<td>undef</td>
<td>undef</td>
</tr>
<tr>
<td>c1</td>
<td>undef</td>
<td>NAC</td>
</tr>
<tr>
<td>NAC</td>
<td>c2</td>
<td>NAC</td>
</tr>
</tbody>
</table>

- Use: \( x \leq y \) implies \( f(x) \leq f(y) \) to check if framework is monotone
  - \([v_1, v_2, \ldots] \leq [v_1', v_2', \ldots], f([v_1, v_2, \ldots]) \leq f([v_1', v_2', \ldots]) \)
**Distributive?**

\[
\begin{align*}
x &= 2 \\
y &= 3 \\
z &= x + y
\end{align*}
\begin{align*}
x &= 3 \\
y &= 2 \\
z &= x + y
\end{align*}
\]

**Summary of Constant Propagation**

- A useful optimization
- Illustrates:
  - abstract execution
  - an infinite semi-lattice
  - a non-distributive problem

**Other Optimizations**

- **Copy Propagation:**

- **Dead Code Elimination:**