Lecture 7

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

I. Available Expressions Analysis

II. Eliminating CSEs

III. Constant Propagation/Folding

[ALSU 9.2.6, 9.4]

Review: A Check List for Data Flow Problems

- **Semi-lattice**
  - set of values \( V \)
  - meet operator \( \land \)
  - Top \( T \)
  - finite descending chain?

- **Transfer functions**
  - function of a basic block \( f: V \rightarrow V \)
  - closed under composition
  - meet-over-paths MOP
  - monotone
  - distributive?

For each node \( n \): \( \text{MOP}(n) = \land f_p(T) \), for all paths \( p \), reaching \( n \)

If data flow framework is **monotone** (i.e., \( x \leq y \) implies \( f(x) \leq f(y) \)) then if the algorithm converges, \( \text{IN}[n] \leq \text{MOP}[b] \)

If data flow framework is **distributive** (i.e., \( f(x \land y) = f(x) \land f(y) \)) then if the algorithm converges, \( \text{IN}[n] = \text{MOP}[b] \)

Example: MOP considers more paths than Ideal

### Ideal: Considers only 2 paths
- B1-B2-B4-B6-B7 (i.e., \( x=1 \))
- B1-B3-B4-B5-B7 (i.e., \( x=0 \))

### MOP: Also considers unexecuted paths
- B1-B2-B4-B5-B7
- B1-B3-B4-B6-B7

Assume: B2 & B3 do not update x
**Review: A Check List for Data Flow Problems**

- **Semi-lattice**
  - set of values $V$
  - meet operator $\lor$
  - Top $T$
  - finite descending chain?

- **Transfer functions**
  - function of a basic block $f: V \rightarrow V$
  - closed under composition
  - meet-over-paths MOP
  - monotone
  - distributive?

- **Algorithm**
  - initialization step (entry/exit, other nodes)
  - visit order: rPostOrder
  - depth of the graph

**Global Common Subexpressions**

- **Availability of an expression $E$ at point $P$**
  - DEFINITION: Along every path to $P$ in the flow graph:
    - $E$ must be evaluated at least once
    - no variables in $E$ redefined after the last evaluation
  - Observation: $E$ may have different values on different paths (e.g., $x+y$ above)

**Available Expressions Example**

- Is $4i$ available at this point?

**Formulating the Problem**

- **Domain:**
  - a bit vector, with a bit for each **textually unique** expression in the program
  - Forward or Backward? Forward
  - Lattice Elements? All bit vectors of given length
  - Meet Operator? Elementwise-min
  - Partial Ordering
  - Top?
  - Bottom?
  - Boundary condition: entry/exit node?
  - Initialization for iterative algorithm?

- **Meet Operator:**
  - Elementwise-min

- **Partial Ordering:**
  - Top?
  - Bottom?
  - Boundary condition: entry/exit node?
  - Initialization for iterative algorithm?

- **Meet Operator:**
  - Intersection
Transfer Functions

• Can use the same equation as reaching definitions
  • \( \text{out}(b) = \text{gen}(b) \cup (\text{in}(b) - \text{kill}(b)) \)

• 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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Available Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a = b + c )</td>
<td>( { } )</td>
</tr>
<tr>
<td>( b = a - d )</td>
<td>( { b + c } )</td>
</tr>
<tr>
<td>( c = b + d )</td>
<td>( { a-d } )</td>
</tr>
<tr>
<td>( d = a - d )</td>
<td>( { } )</td>
</tr>
</tbody>
</table>

Initialization for Interior Nodes

\[ \text{out}(b) = \text{gen}(b) \cup (\text{in}(b) - \text{kill}(b)) \]

T = \{e1, e2\}
\[ \{e1\} \quad \{e2\} \quad \{\} \]
Meet Operator: Intersection

- What if initialize \( \text{out}(B2) = \{\} \)? Incorrect: in(B2)=c
- What if initialize \( \text{out}(B2) = \{\} \)? Correct: in(B2)=out(B1)
- Initialize \( \text{out}(b) = \{\} \) for all interior b

Composing Transfer Functions

• Derive the transfer function for an entire block

\[ \text{in}1 \]
\[ \text{out}1 = \text{gen}1 \cup (\text{in}1 - \text{kill}1) = \text{in}2 \]
\[ \times \]
\[ \text{out}2 = \text{gen}2 \cup (\text{in}2 - \text{kill}2) \]

• Since \( \text{out}1 = \text{in}2 \) we can simplify:
  - \( \text{out}2 = \text{gen}2 \cup (\text{gen}1 \cup (\text{in}1 - \text{kill}1)) - \text{kill}2 \)
  - \( \text{out}2 = \text{gen}2 \cup (\text{gen}1 - \text{kill}2) \cup (\text{in}1 - (\text{kill}1 \cup \text{kill}2)) \)
  - \( \text{out}2 = \text{gen}2 \cup (\text{gen}1 - \text{kill}2) \cup (\text{in}1 - (\text{kill}1 \cup \text{kill}2)) \)

• Result
  - \( \text{gen} = \text{gen}2 \cup (\text{gen}1 - \text{kill}2) \)
  - \( \text{kill} = \text{kill}2 \cup (\text{kill}1 - \text{gen}2) \)

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
**Review: Value Numbering**

\[
\begin{align*}
    a &= b + c \\
    b &= a - d \\
    c &= b + c \\
    d &= a - d
\end{align*}
\]

**Example Revisited**

\[
\begin{align*}
    t_1 &= b + c \\
    a &= t_1 \\
    t_2 &= t_1 - d \\
    b &= t_2 \\
    t_3 &= t_2 + c \\
    c &= t_3 \\
    t_4 &= t_2 + c \\
    c &= t_3 \\
    d &= t_2
\end{align*}
\]

**Limitation: Textually Identical Expressions**

- Commutative operations:

\[
\begin{align*}
    \text{add } t_1 &= x, y \\
    \text{add } t_2 &= y, x \\
    \text{add } t_3 &= x, y
\end{align*}
\]

- sort the operands

**Further Improvements**

- Examples:
  - Expressions with more than two operands:

\[
\begin{align*}
    \text{add } t_1 &= x, y \\
    \text{add } t_2 &= t_1, z \\
    \text{add } t_3 &= y, x \\
    \text{add } t_4 &= t_3, x \\
    \text{add } t_5 &= t_3, x \\
    \text{add } t_6 &= t_5, z
\end{align*}
\]

- Textually different expressions may be equivalent:

\[
\begin{align*}
    \text{add } t_1 &= x, y \\
    \text{beq } t_1, t_2, l_1 \\
    \text{cpy } z &= x \\
    \text{add } t_3 &= z, y
\end{align*}
\]

Use multiple passes of GCSE combined with copy propagation
### Summary

<table>
<thead>
<tr>
<th>Reaching Definitions</th>
<th>Available Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Sets of definitions</td>
</tr>
<tr>
<td></td>
<td>Sets of expressions</td>
</tr>
<tr>
<td>Transfer function</td>
<td>Generate U Propagate</td>
</tr>
<tr>
<td>direction of function</td>
<td>forward: out(b) = f(in(b))</td>
</tr>
<tr>
<td>Generate</td>
<td>Gen_n: expressions evaluated</td>
</tr>
<tr>
<td>Propagate</td>
<td>n(b)-KILL_n: definitions killed</td>
</tr>
<tr>
<td>Meet operation</td>
<td>U (in(b)+out(predecessors))</td>
</tr>
<tr>
<td>Initialization</td>
<td>out(entry) = Ø</td>
</tr>
<tr>
<td></td>
<td>out(b) = Ø</td>
</tr>
</tbody>
</table>

### 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?

```
\( \text{e} = 1 \)
\( \text{e, x, m are each a constant value} \)
\( \text{x} = 2 \)
\( \text{m} = \text{x} + \text{e} \)
\( \text{e} = 3 \)
```

```
\( \text{e, x, m are each a constant value} \)
\( \text{e} = 1 \)
\( \text{m} = \text{x} + \text{e} \)
\( \text{e} = 3 \)
```

### 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>c_2</td>
<td>c_2</td>
<td>c_2</td>
</tr>
<tr>
<td>NAC</td>
<td>NAC</td>
<td>NAC</td>
</tr>
<tr>
<td>c_1</td>
<td>undef</td>
<td>c_1</td>
</tr>
<tr>
<td>c_2</td>
<td>c_2</td>
<td>c_2 (if c_1 \leq c_2)</td>
</tr>
<tr>
<td>NAC</td>
<td>NAC</td>
<td>NAC</td>
</tr>
<tr>
<td>NAC</td>
<td>NAC</td>
<td>NAC</td>
</tr>
</tbody>
</table>
```

- Note: \( \text{undef} \land c_2 = c_2 \)
Example

\[
x = \text{UNDEF} \\
x = 2 \\
x = 2 \\
x = \text{UNDEF} \\
p = x
\]

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 \( b \)
• \( \text{OUT}[\text{entry}, x] = \text{undef} \), 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
  • \( \text{OUT}[b, x] = \text{IN}[b, x] \), if \( x \neq x_1 \)

<table>
<thead>
<tr>
<th>( \text{IN}[b, x_1] )</th>
<th>( \text{IN}[b, x_2] )</th>
<th>( \text{OUT}[b, x_3] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{undef} )</td>
<td>( \text{undef} )</td>
<td>( \text{undef} )</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>( c_2 )</td>
<td>( c_1 + c_2 )</td>
</tr>
<tr>
<td>( \text{NAC} )</td>
<td>( \text{NAC} )</td>
<td>( \text{NAC} )</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>( c_2 )</td>
<td>( NAC )</td>
</tr>
<tr>
<td>( \text{NAC} )</td>
<td>( \text{NAC} )</td>
<td>( \text{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?

\[
x = 2 \\
y = 3 \\
x = 3 \\
y = 2 \\
z = x + y
\]

• Not Distributive
• Iterative solution is not precise!
  – it is also not wrong
  – it is conservative
**Summary of Constant Propagation**

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

**Monday’s Class**

- Static Single Assignment (SSA)  [ALSU 6.2.4]