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

Reading: 9.2.6, 9.4
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
  
  - Observations: \( E \) may have different values on different paths

```
add t1 = x, y
add t2 = c, d

dlct3 = 0
cpyx = t3
add t4 = x, y
cpy m = t4

sub t5 = a, b
ldct6 = -1
cpy c = t6

sub t7 = a, b
cpy m = t7
add t8 = x, y
add t9 = c, d
```
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
  - \( \text{out}[b] = \text{gen}[b] \cup (\text{in}[b] - \text{kill}[b]) \)

- 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
**Composing Transfer Functions**

- Derive the transfer function for an entire block

\[ \text{in1} \]

\[ \downarrow \]

\[ 1 \]

\[ \downarrow \]

\[ \text{out1} = \text{gen1} \lor (\text{in1} - \text{kill1}) = \text{in2} \]

\[ \downarrow \]

\[ 2 \]

\[ \downarrow \]

\[ \text{out2} = \text{gen2} \lor (\text{in2} - \text{kill2}) \]

- Since out1 = in2 we can simplify:

  - out2 = gen2 \lor ((gen1 \lor (in1 - kill1)) - kill2)
  - out2 = gen2 \lor (gen1 - kill2) \lor (in1 - (kill1 \lor kill2))
  - out2 = gen2 \lor (gen1 - kill2) \lor (in1 - (kill2 \lor (kill1 - gen2)))

- Result

  - gen = gen2 \lor (gen1 - kill2)
  - kill = kill2 \lor (kill1 - 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
Example Revisited

```
add t1 = x, y
add t2 = c, d

ldc t3 = 0
cpy x = t3
add t4 = x, y
cpy m = t4

sub t5 = a, b
ldc t6 = -1
cpy c = t6

sub t7 = a, b
cpy m = t7
add t8 = x, y
add t9 = c, d
```
III. Limitation: Textually Identical Expressions

- Commutative operations

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

• Examples
  – Expressions with more than two operands

```
add t1 = x, y
add t2 = t1, z
add t3 = y, x
add t4 = t3, z
add t5 = x, y
add t6 = t5, z
```

  – Textually different expressions may be equivalent

```
add t1 = x, y
beq t1, t2, L1
cpy z = x
add t3 = z, y
```
Another Example

\[
\begin{align*}
x &= 1 \\
y &= 1
\end{align*}
\]

\[
\begin{align*}
x &= x + 1 \\
y &= y + 1 \\
x &= x + 1 \\
y &= y + 1
\end{align*}
\]
## Summary

<table>
<thead>
<tr>
<th></th>
<th>Reaching Definitions</th>
<th>Available Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Sets of definitions</td>
<td>Sets of expressions</td>
</tr>
<tr>
<td>Transfer function $f_b(x)$</td>
<td>Generate U Propagate</td>
<td></td>
</tr>
<tr>
<td>direction of function</td>
<td>forward: $\text{out}[b] = f_b(\text{in}[b])$</td>
<td>forward: $\text{out}[b] = f_b(\text{in}[b])$</td>
</tr>
<tr>
<td>Generate</td>
<td>$\text{Gen}_b$: exposed definitions</td>
<td>$\text{Gen}_b$: expressions evaluated</td>
</tr>
<tr>
<td>Propagate</td>
<td>$\text{in}[b]$: definitions killed</td>
<td>$\text{in}[b]$: definitions killed</td>
</tr>
<tr>
<td>Meet operation</td>
<td>$U (\text{in}[b] = U \text{out}[\text{predecessors}])$</td>
<td>$\cap (\text{in}[b] = \cap \text{out}[\text{predecessors}])$</td>
</tr>
<tr>
<td>Initialization</td>
<td>$\text{out}[\text{entry}] = \emptyset$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{out}[b] = \emptyset$</td>
<td>$\text{out}[\text{entry}] = \emptyset$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{out}[b] = \text{all expressions}$</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?

\[
\begin{align*}
  e &= 1 \\
  x &= 2 \\
  m &= x + e \\
  e &= 3 \\
  p &= e + 4
\end{align*}
\]
Semi-lattice Diagram

- Finite domain?
- Finite height?
### Equivalent Definition

- **Meet Operation:**

<table>
<thead>
<tr>
<th>v1</th>
<th>v2</th>
<th>v1 &amp; v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>undef</td>
<td>undef</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c₂</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAC</td>
<td></td>
</tr>
<tr>
<td>c₁</td>
<td>undef</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c₂</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAC</td>
<td></td>
</tr>
<tr>
<td>NAC</td>
<td>undef</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c₂</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAC</td>
<td></td>
</tr>
</tbody>
</table>

- **Note:** `undef \& c₂ = c₂`!
Example

\[ x = 2 \]

\[ p = x \]
Transfer Function

• Assume a basic block has only 1 instruction

• Let $IN[b,x],\ OUT[b,x]$
  
  – be the information for variable $x$ at entry and exit of basic block $b$

• $OUT[\text{entry, } x] = \text{undef}$, for all $x$.

• Non-assignment instructions: $OUT[b,x] = IN[b,x]$

• 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 $x \neq x_3$

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

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

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

\[ z = x + y \]
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:**