Lecture 4

Advanced Topics: Constant Propagation Speed

I. Constant Propagation
II. Efficiency of Data Flow Analysis

Reading: Chapter 9.4
I. 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 ∧ 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`!
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 $x \neq x_3$

<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></td>
</tr>
<tr>
<td></td>
<td>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>NAC</td>
<td></td>
</tr>
<tr>
<td>NAC</td>
<td>undef</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$c_2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAC</td>
<td></td>
</tr>
</tbody>
</table>

1. Hold $x_1$ constant, lower $x_2$, show results do not rise.
2. Hold $x_2$ constant, lower $x_1$, show results do not rise.
3. Combine above: to prove monotonicity.

- **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?

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

- A useful optimization
- Illustrates:
  - abstract execution
  - an infinite semi-lattice
  - a non-distributive problem
II. Efficiency of Iterative Data Flow

- Assume forward data flow for this discussion
- Speed of convergence depends on the ordering of nodes

• How about:

I. A → B → C → D → E
II. A → D → E → B → C
Reverse Post-order: A → B → C → D → E

Pre-order: (1) A B C D E exit
Post-order: exit E D C B A
**Depth-first Ordering: Reverse Postorder**

- **Preorder traversal**: visit the parent before its children
- **Postorder traversal**: visit the children then the parent
- **Preferred ordering**: reverse postorder
- **Intuitively**
  - depth first postorder visits the farthest node as early as possible
  - reverse postorder delays visiting farthest node
“Reverse Post-Order” Iterative Algorithm

input: control flow graph CFG = (N, E, Entry, Exit)

// Boundary condition
OUT[Entry] = \emptyset

// Initialization for iterative algorithm
For each basic block B other than Entry
OUT[B] = \emptyset

// iterate
While (changes to any OUT occur) {
For each basic block B other than Entry in reverse post order {
IN[B] = \cup (OUT[p]), for all predecessors p of B
OUT[B] = f_B(IN[B]) // OUT[B]=gen[B]\cup(IN[B]-kill[B])
}
}
Consideration of Speed of Convergence

- Does it matter if we go around the same cycle multiple times?
- Reachability problems: “Does a path exist?”
  - Reaching definitions, liveness
  - Does not matter how many times we go around cycles
- Traversing cycles can make a difference: constant propagation

\[
\begin{align*}
a &= b \\
b &= c \\
c &= 1
\end{align*}
\]
Speed of Convergence

• If cycles do not add info:
  – Labeling nodes in a path by their reverse postorder rank:
    1 → 4 → 5 → 7 → 2 → 4 ...
  – info flows down nodes of increasing reverse postorder rank in 1 pass
• Loop depth = max. # of “retreating edges” in any acyclic path
• Maximum # iterations in data flow algorithm = Loop depth+2
  (2 is necessary even if there are no cycles)

Knuth’s experiments show: average loop depth in real programs = 2.75
Summary

• **Constant propagation**
  – abstract execution
  – an infinite semi-lattice
  – a non-distributive framework

• **Convergence**
  – Reverse postorder iterative algorithm
    • Faster than worklist algorithm for reachability-based data problems
    • The typical loop depth is low