Lecture 4

More on Data Flow:
Constant Propagation
Control Flow: Speed, Loops

I  Constant Propagation
II  Efficiency of Data Flow Analysis
III  Algorithm to find loops

Reading: Chapter 9.4, 9.6
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>undef</td>
</tr>
<tr>
<td></td>
<td>c₂</td>
<td>c₂</td>
</tr>
<tr>
<td></td>
<td>NAC</td>
<td>NAC</td>
</tr>
<tr>
<td>c₁</td>
<td>undef</td>
<td>c₁</td>
</tr>
</tbody>
</table>
|       | c₂   | c₁, if c₁ = c₂  
|       |      | NAC otherwise |
|       | NAC  | NAC     |
| NAC  | undef | NAC     |
|       | c₂   | NAC     |
|       | NAC  | NAC     |

- **Note:** undef ∧ c₂ = c₂!
Example

\[ x = 2 \]

\[ p = x \]
Transfer Functions

• 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>

- 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?

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

\[ z = x + y \]
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

II. A
    D
    E
    B
    C
    exit
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] = ∅

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

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

Does it matter if we go around the same cycle multiple times?

- **Cycles do not make a difference:**
  - reaching definitions, liveness

- **Cycles 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
III. What is a Loop?

- **Goal:**
  - Define a loop in graph-theoretic terms (control flow graph)
  - Not sensitive to input syntax, a uniform treatment for all loops: DO, while, goto’s

- **Informally: A “natural” loop has**
  - edges that form at least a cycle
  - a single entry point
Dominators

- Node \( d \) dominates node \( n \) in a graph \((d \ dom n)\) if every path from the start node to \( n \) goes through \( d \)
  - a node dominates itself

- Immediate dominance:
  \[ d \ \idom\ n : d \ dom n, d \neq n, \neg \exists m \text{ s.t. } d \ dom m \text{ and } m \ dom n \]
- Immediate dominance relationships form a tree
Finding Dominators

- **Definition**
  - Node $d$ dominates node $n$ in a graph ($d \text{ dom } n$) if every path from the start node to $n$ goes through $d$

- **Formulated as a MOP problem**
  - node $d$ lies on all possible paths reaching node $n \Rightarrow d \text{ dom } n$
    - Direction:
    - Values:
    - Meet operator:
    - Top:
    - Bottom:
    - Boundary condition: start/exit node =
    - Finite descending chains only?
    - Transfer function:

- **Speed:**
  - With reverse postorder, solution to most flow graphs (reducible flow graphs) found in 1 pass
Definition of Natural Loops

• Single entry-point: **header** \((d)\)
a header dominates all nodes in the loop

• A **back edge** \((n \rightarrow d)\) in a flow graph is
an edge whose destination dominates its source \((d \text{ dom } n)\)

• The **natural loop of a back edge** \((n \rightarrow d)\) is
\(d + \{ \text{nodes that can reach } n \text{ without going through } d \}\).
Graph Edges

- **Depth-first spanning tree**
  - Edges traversed in a depth-first search of a graph form a depth-first spanning tree

- **Categorizing edges in graph**
  - Advancing edges: from ancestor to proper descendant
  - Retreating edges: from descendant to ancestor (not necessarily proper)
  - Cross edges: all other edges
Back Edges

• Definition
  • Back edge: $n \rightarrow d$, $d$ dom $n$

• Relationships between graph edges and back edges
  • a back edge must be a retreating edge
dominator $\Rightarrow$ visit $d$ before $n$, $n$ must be a descendant of $d$
  • a retreating edge is not necessarily a back edge

• Most programs (all structured code, and most GOTO programs)
  • retreating edges = back edges
Constructing Natural Loops

• The *natural loop of a back edge* \((n \rightarrow d)\) is 
  \(d + \{ \text{nodes that can reach } n \text{ without going through } d \} \).

• Remove \(d\) from the flow graph, find all predecessors of \(n\)

• Example

```
1 ——— 2 ——— 3 ——— 4 ——— 5 ——— 6 ——— 7 ——— 8
```

1 ——— 2

1 ——— 2 ——— 3

1 ——— 2 ——— 3 ——— 4
Inner Loops

- **If two loops do not have the same header**
  - they are either disjoint, or
  - one is entirely contained (nested within) the other
    -- inner loop, one that contains no other loop.

- **If two loops share the same header**
  - Hard to tell which is the inner loop
  - Combine as one
Summary

• Constant propagation

• Introduced the reverse postorder iterative algorithm

• Define loops in graph theoretic terms

• Definitions and algorithms for
  • Dominators
  • Back edges
  • Natural loops