Announcement
- HW2 is due Monday! I will not accept late HW2 turnins

Idea
- Eliminate redundant operations in the dynamic execution of instructions

How do redundancies arise?
- Loop invariant code (e.g., index calculation for arrays)
- Sequence of similar operations (e.g., method lookup)
- Same value be generated in multiple places in the code

Types of reuse optimization
- Value numbering
- Common subexpression elimination
- Partial redundancy elimination

Local Value Numbering

Idea
- Each variable, expression, and constant is assigned a unique number
- When we encounter a variable, expression or constant, see if it’s already been assigned a number
- If so, use the value for that number
- If not, assign a new number
- Same number ⇒ same value

Example
a := b + c
d := b
e := d + c

b → #1
c → #2
b + c is #1 + #2 → #3
a → #1
a + c is #1 + #2 → #3
d → #3
e ← #3

t := b + c

b → #1
c → #2
b + c is #1 + #2 → #3
da ← #2
d + c is #1 + #2 → #3
e ← #3

Local Value Numbering (cont)

Temporaries may be necessary

| a := b + c | b → #1 |
| a := b    | c → #2 |
| d := a + c | b + c is #1 + #2 → #3 |
| t := b + c | a → #1 |
| a := b    | a + c is #1 + #2 → #3 |
| d := b + c | d → #3 |

Global Value Numbering

How do we handle control flow?

w := 5
x := 5
w := 8
x := 8
y := w + 1
z := x + 1
w := #1
x := #1
z := #1
y := #1
w := #2
x := #2
**Global Value Numbering (cont)**

Idea [Alpern, Wegman, and Zadeck 1988]
- Partition program variables into congruence classes
- All variables in a particular congruence class have the same value
- SSA form is helpful

**Approaches to computing congruence classes**
- Pessimistic
  - Assume no variables are congruent (start with $n$ classes)
  - Iteratively coalesce classes that are determined to be congruent
- Optimistic
  - Assume all variables are congruent (start with one class)
  - Iteratively partition variables that contradict assumption
  - Slower but better results

**Role of SSA Form**

SSA form is helpful
- Allows us to avoid data-flow analysis
- Variables correspond to values

\[
a = b \quad \cdots \quad a = c \quad \cdots \quad a = d
\]

\[
a_1 = b \quad \cdots \quad a_2 = c \quad \cdots \quad a_3 = d
\]

Congruence classes:
\[
\{a_2, b\}, \{a_3, c\}, \{a_1, d\}
\]

**Basis**

Idea
- If $x$ and $y$ are congruent then $f(x)$ and $f(y)$ are congruent

\[
x = f(a, b) \quad y = f(t_a, t_b)
\]

- Use this fact to combine (pessimistic) or split (optimistic) classes

**Problem**
- This is not true for $\phi$-functions

\[
a_1 = x \quad s_1 = y_1 \quad b_1 = x_1 \quad b_2 = y_1
\]

\[
a_2 = \phi(a_1, a_2) \quad b_3 = \phi(b_1, b_2)
\]

**Solution**
- Label $\phi$-functions with join point

**Pessimistic Global Value Numbering**

Idea
- Initially each variable is in its own congruence class
- Consider each assignment statement $s$ (reverse postorder in CFG)
- Update LHS value number with hash of RHS
- Identical value number $\Rightarrow$ congruence

**Why reverse postorder?**
- Ensures that when we consider an assignment statement, we have already considered definitions that reach the RHS operands

*Postorder: d, c, e, b, f, a*
Algorithm

for each assignment of the form: “x = f(a, b)”
	ValNum[x] ← UniqueValue() // same for a and b
for each assignment of the form: “x = f(a, b)” (in reverse postorder)
	ValNum[x] ← Hash(f ⊕ ValNum[a] ⊕ ValNum[b])

Snag!

Problem
- Our algorithm assumes that we consider operands before variables that depend upon it
- Can’t deal with code containing loops!

Solution
- Ignore back edges
- Make conservative (worst case) assumption for previously unseen variable (i.e., assume its in its own congruence class)

Optimistic Global Value Numbering

Idea
- Initially all variables in one congruence class
- Split congruence classes when evidence of non-congruence arises
  - Variables that are computed using different functions
  - Variables that are computed using functions with non-congruent operands

Splitting

Initially
- Variables computed using the same function are placed in the same class
  - x₁ = f(a₁, b₁)
  - y₁ = f(c₁, d₁)
  - z₁ = f(e₁, f₁)

Iteratively
- Split classes when corresponding operands are in different classes
- Example: a₁ and c₁ are congruent, but e₁ is congruent to neither

---

![Diagram showing value numbering algorithm and optimistic global value numbering concepts.](image-url)
Splitting (cont)

Definitions
- Suppose P and Q are sets representing congruence classes
- Q splits P for each i into two sets
  - P \ Q contains variables in P whose i\textsuperscript{th} operand is in Q
  - P / Q contains variables in P whose i\textsuperscript{th} operand is not in Q
- Q properly splits P if neither resulting set is empty

\[
x_1 = f(a, b) \quad P \\
y_1 = f(c, d) \quad P \\
z_1 = f(e, f)
\]

Algorithm

\[
\text{worklist} \leftarrow \emptyset \\
\text{for each} \quad \text{function } f \quad \text{C}_f \leftarrow \emptyset \\
\text{for each} \quad \text{assignment of the form } "x = f(a, b)" \\
\quad \text{C}_f \leftarrow \text{C}_f \cup \{ x \} \\
\text{worklist} \leftarrow \text{worklist} \cup \{ \text{C}_f \} \\
\text{CC} \leftarrow \text{CC} \cup \{ \text{C}_f \} \\
\text{while} \quad \text{worklist} \neq \emptyset \\
\quad \text{Delete some } D \text{ from worklist} \\
\quad \text{for each} \quad \text{class } C \text{ properly split by } D \text{ (at operand } i) \\
\quad \quad \text{CC} \leftarrow \text{CC} \setminus \text{C}_f \\
\quad \text{worklist} \leftarrow \text{worklist} \setminus \text{C}_f \\
\quad \text{Create new congruence classes } C_j \leftarrow \{ C \setminus i D \} \text{ and } C_k \leftarrow \{ C \setminus i D \} \\
\quad \text{CC} \leftarrow \text{CC} \cup C_j \cup C_k \\
\text{worklist} \leftarrow \text{worklist} \cup C_j \cup C_k \\
\text{Note: see paper for optimization}
\]

Example

SSA code

\[
\begin{align*}
x_0 &= 1 \\
y_0 &= 2 \\
x_1 &= x_0 + 1 \\
y_1 &= y_0 + 1 \\
z_1 &= x_0 + 1
\end{align*}
\]

Congruence classes

\[
\begin{array}{ll}
S_0 & \{ x_0 \} \\
S_1 & \{ y_0 \} \\
S_2 & \{ x_0, y_1, z_1 \} \\
S_3 & \{ x_1, z_1 \} \\
S_4 & \{ y_1 \}
\end{array}
\]

Worklist: \text{S}_0, \text{S}_1, \text{S}_2, \text{S}_3, \text{S}_4

<table>
<thead>
<tr>
<th>S_0 psplit S_1?</th>
<th>S_0 psplit S_2?</th>
<th>S_0 psplit S_3?</th>
<th>S_0 psplit S_4?</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>

S_2 \setminus S_0 = \{ x_1, z_1 \} = S_3
S_2 \setminus S_1 = \{ y_1 \} = S_4

Comparing Optimistic and Pessimistic

Differences
- Handling of loops
- Pessimistic makes worst-case assumptions on back edges
- Optimistic requires actual contradiction to split classes
Role of SSA

Single global result
- Single def reaches each use
- No data (flow value) at each point

No data flow analysis
- Optimistic: Iterate over congruence classes, not CFG nodes
- Pessimistic: Visit each assignment once

$\phi$-functions
- Make data-flow merging explicit
- Treat like normal functions

Next Time

Lecture
- Midterm Review
- Send questions you would like answered at the Midterm Review