

## Reuse Optimization

### 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

CS553 Lecture

Value Numbering

2

## 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  $\Rightarrow$  same value

### Example

a	:=	b	+	c		b $\rightarrow$ #1 #3
d	:=	b				c $\rightarrow$ #2
b	:=	a				b + c is #1 + #2 $\rightarrow$ #3
e	:=	d	+	c	a	a $\rightarrow$ #3
						d $\rightarrow$ #1
						d + c is #1 + #2 $\rightarrow$ #3
						e $\rightarrow$ #3

CS553 Lecture

Value Numbering

3

## Local Value Numbering (cont)

### Temporaries may be necessary

a	:=	b	+	c	b $\rightarrow$ #1
a	:=	b			c $\rightarrow$ #2
d	:=	a	+	c	b + c is #1 + #2 $\rightarrow$ #3
					a $\rightarrow$ #3 #1
					a + c is #1 + #2 $\rightarrow$ #3
					d $\rightarrow$ #3

t	:=	b	+	c	b $\rightarrow$ #1
a	:=	b			c $\rightarrow$ #2
d	:=	b	+	c	b + c is #1 + #2 $\rightarrow$ #3
					t $\rightarrow$ #3
					a $\rightarrow$ #1
					a + c is #1 + #2 $\rightarrow$ #3
					d $\rightarrow$ #3

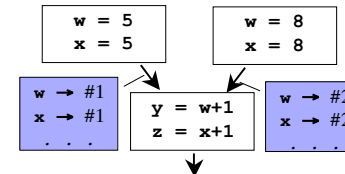
CS553 Lecture

Value Numbering

4

## Global Value Numbering

### How do we handle control flow?



CS553 Lecture

Value Numbering

5

## 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

CS553 Lecture

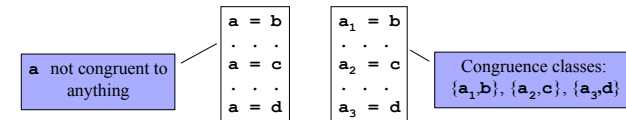
Value Numbering

6

## Role of SSA Form

### SSA form is helpful

- Allows us to avoid data-flow analysis
- Variables correspond to values



CS553 Lecture

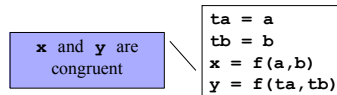
Value Numbering

7

## Basis

### Idea

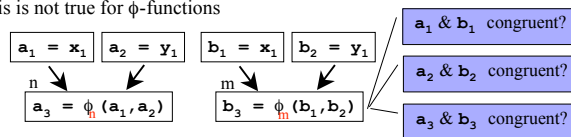
- If  $x$  and  $y$  are congruent then  $f(x)$  and  $f(y)$  are congruent



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

### Problem

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



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

CS553 Lecture

Value Numbering

8

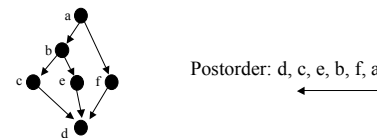
## Pessimistic Global Value Numbering

### 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



CS553 Lecture

Value Numbering

9

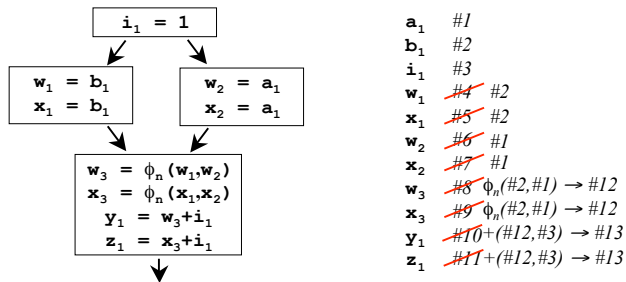
## Algorithm

for each assignment of the form: " $x = f(a, b)$ "

ValNum[x]  $\leftarrow$  UniqueValue() // same for a and b

for each assignment of the form: " $x = f(a, b)$ " (in reverse postorder)

ValNum[x]  $\leftarrow$  Hash( $f \oplus$  ValNum[a]  $\oplus$  ValNum[b])



CS553 Lecture

Value Numbering

10

## 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)

CS553 Lecture

Value Numbering

11

## 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

CS553 Lecture

Value Numbering

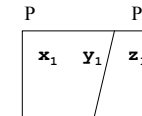
12

## Splitting

### Initially

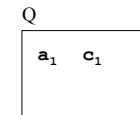
- Variables computed using the same function are placed in the same class

$x_1 = f(a_1, b_1)$   
 $y_1 = f(c_1, d_1)$   
 $z_1 = f(e_1, f_1)$



### Iteratively

- *Split* classes when corresponding operands are in different classes
- Example:  $a_1$  and  $c_1$  are congruent, but  $e_1$  is congruent to neither



CS553 Lecture

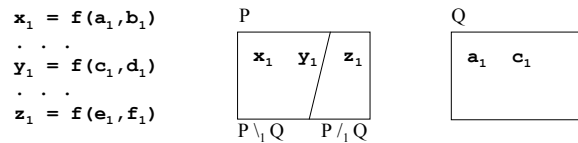
Value Numbering

13

## Splitting (cont)

### Definitions

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



CS553 Lecture

Value Numbering

14

## Algorithm

```

worklist ← ∅
for each function f
  Cf ← ∅
  for each assignment of the form "x = f(a,b)"
    Cf ← Cf ∪ {x}
  worklist ← worklist ∪ {Cf}
  CC ← CC ∪ {Cf}
while worklist ≠ ∅
  Delete some D from worklist
  for each class C properly split by D (at operand i)
    CC ← CC - C
    worklist ← worklist - C
    Create new congruence classes Cj ← {C \ D} and Ck ← {C / D}
    CC ← CC ∪ Cj ∪ Ck
    worklist ← worklist ∪ Cj ∪ Ck
    
```

Note: see paper for optimization

CS553 Lecture

Value Numbering

15

## Example

SSA code	Congruence classes
$\mathbf{x}_0 = 1$	$S_0 = \{\mathbf{x}_0\}$
$\mathbf{y}_0 = 2$	$S_1 = \{\mathbf{y}_0\}$
$\mathbf{x}_1 = \mathbf{x}_0 + 1$	<del><math>S_2 = \{\mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1\}</math></del>
$\mathbf{y}_1 = \mathbf{y}_0 + 1$	$S_3 = \{\mathbf{x}_1, \mathbf{z}_1\}$
$\mathbf{z}_1 = \mathbf{x}_0 + 1$	$S_4 = \{\mathbf{y}_1\}$

Worklist:  ~~$S_0 = \{\mathbf{x}_0\}$~~ ,  ~~$S_1 = \{\mathbf{y}_0\}$~~ ,  ~~$S_2 = \{\mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1\}$~~ ,  ~~$S_3 = \{\mathbf{x}_1, \mathbf{z}_1\}$~~ ,  ~~$S_4 = \{\mathbf{y}_1\}$~~

$S_0$  psplit  $S_0$ ? **no**      $S_0$  psplit  $S_1$ ? **no**      $S_0$  psplit  $S_2$ ? **yes!**

$S_2 \setminus_i S_0 = \{\mathbf{x}_1, \mathbf{z}_1\} = S_3$

$S_2 /_i S_0 = \{\mathbf{y}_1\} = S_4$

CS553 Lecture

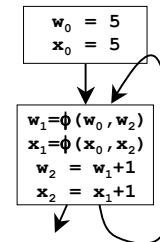
Value Numbering

16

## Comparing Optimistic and Pessimistic

### Differences

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



CS553 Lecture

Value Numbering

17

## 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