

LLVM Compiler Infrastructure

Source: "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation" by Lattner and Adiv

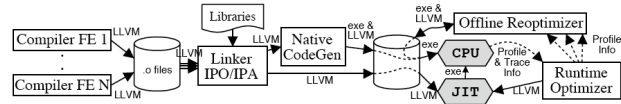


Figure 4: LLVM system architecture diagram

Goals

- lifelong analysis and optimization
- modular compiler components

IR is low level, control-flow graph and SSA

- language-independent types: 8-26 bit ints, float, double, pointers, arrays, structures, functions

Status

- Apple is paying main developer (Chris Lattner) to continue development
- built OpenGL JIT compiler with it in two weeks

CS553 Lecture

Value Numbering

1

Reuse Optimization

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

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

CS553 Lecture

Value Numbering

3

Local Value Numbering (cont)

Temporaries may be necessary

```
a := b + c
a := b
d := a + c
```

```
b → #1
c → #2
b + c is #1 + #2 → #3
a → #3 #1
a + c is #1 + #2 → #3
d → #3
```

```
t := b + c
a := b
d := b + c t
```

```
b → #1
c → #2
b + c is #1 + #2 → #3
t → #3
a → #1
a + c is #1 + #2 → #3
d → #3
```

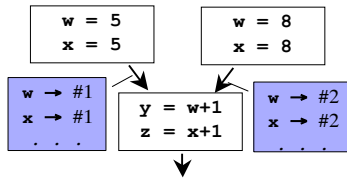
CS553 Lecture

Value Numbering

4

Global Value Numbering

How do we handle control flow?



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

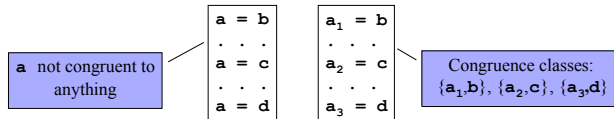
Approaches to computing congruence classes

- 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

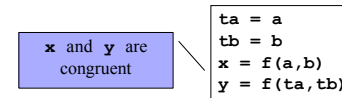
- Allows us to avoid data-flow analysis
- Variables correspond to values



Basis

Idea

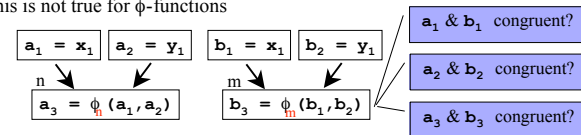
- If x and y are congruent then $f(x)$ and $f(y)$ are congruent



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

Problem

- This is not true for ϕ -functions



Solution: Label ϕ -functions with join point

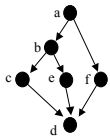
Pessimistic Global Value Numbering

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

CS553 Lecture

Value Numbering

9

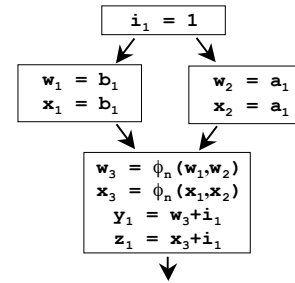
Algorithm

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])



a_1 #1
 b_1 #2
 i_1 #3
 w_1 ~~#4~~ #2
 x_1 ~~#5~~ #2
 w_2 ~~#6~~ #1
 x_2 ~~#7~~ #1
 w_3 ~~#8~~ $\phi_n(\#2, \#1) \rightarrow \#12$
 x_3 ~~#9~~ $\phi_n(\#2, \#1) \rightarrow \#12$
 y_1 ~~#10~~ $(\#12, \#3) \rightarrow \#13$
 z_1 ~~#11~~ $(\#12, \#3) \rightarrow \#13$

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

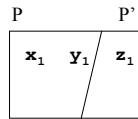
12

Splitting

Initially

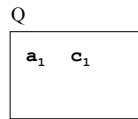
- Variables computed using the same function are placed in the same class

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{f}(\mathbf{a}_1, \mathbf{b}_1) \\ \cdot & \cdot \\ \mathbf{y}_1 &= \mathbf{f}(\mathbf{c}_1, \mathbf{d}_1) \\ \cdot & \cdot \\ \mathbf{z}_1 &= \mathbf{f}(\mathbf{e}_1, \mathbf{f}_1) \end{aligned}$$



Iteratively

- Split* classes when corresponding operands are in different classes
- Example: assume \mathbf{a}_1 and \mathbf{c}_1 are congruent, but \mathbf{e}_1 is congruent to neither



CS553 Lecture

Value Numbering

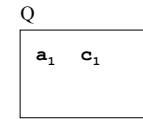
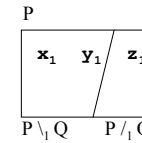
13

Splitting (cont)

Definitions

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

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{f}(\mathbf{a}_1, \mathbf{b}_1) \\ \cdot & \cdot \\ \mathbf{y}_1 &= \mathbf{f}(\mathbf{c}_1, \mathbf{d}_1) \\ \cdot & \cdot \\ \mathbf{z}_1 &= \mathbf{f}(\mathbf{e}_1, \mathbf{f}_1) \end{aligned}$$



CS553 Lecture

Value Numbering

14

Algorithm

```

worklist  $\leftarrow \emptyset$ 
for each function f
   $C_f \leftarrow \emptyset$ 
  for each assignment of the form "x = f(a,b)"
     $C_f \leftarrow C_f \cup \{x\}$ 
  worklist  $\leftarrow$  worklist  $\cup \{C_f\}$ 
   $CC \leftarrow CC \cup \{C_f\}$ 
while worklist  $\neq \emptyset$ 
  Delete some D from worklist
  for each class C properly split by D (at operand i)
     $CC \leftarrow CC - C$ 
    worklist  $\leftarrow$  worklist  $- C$ 
    Create new congruence classes  $C_j \leftarrow \{C \setminus D\}$  and  $C_k \leftarrow \{C /_i D\}$ 
     $CC \leftarrow CC \cup C_j \cup C_k$ 
    worklist  $\leftarrow$  worklist  $\cup C_j \cup C_k$ 
    
```

Note: see paper for optimization

CS553 Lecture

Value Numbering

15

Example

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

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!**

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

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

CS553 Lecture

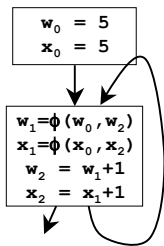
Value Numbering

16

Comparing Optimistic and Pessimistic

Differences

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



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

ϕ -functions

- Make data-flow merging explicit
- Treat like normal functions after subscripting them for each merge point

Next Time

Lecture

- Parallelism and Data Locality, start reading chapter 11 in dragon book

Suggested Exercise

- Use any programming language to write the class declarations for the equivalence class and variable structures described at the bottom of page 6 in the value numbering chapter.
- Instantiate the data structures for the small example programs in Figures 7.2 and 7.6 and perform optimistic value numbering. Remember to convert to SSA first.
- After performing value numbering, how will the program be transformed?
- Now perform pessimistic value numbering on the two examples. Transform the code as well.