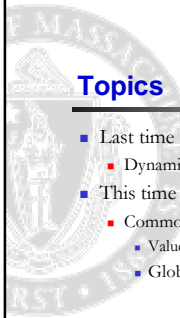


Advanced Compilers
 CMPSCI 710
 Spring 2003
Common Subexpression Elimination
Emery Berger
 University of Massachusetts, Amherst

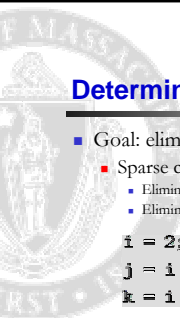
UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



Topics

- Last time
 - Dynamic storage allocation, garbage collection
- This time
 - Common subexpression elimination
 - Value numbering
 - Global CSE

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



Determining Equivalence

- Goal: eliminate redundant computations
 - Sparse conditional constant propagation:
 - Eliminates multiple computations
 - Eliminates unnecessary branches

```

i = 2;
j = i * 2;
k = i + 2;
  
```

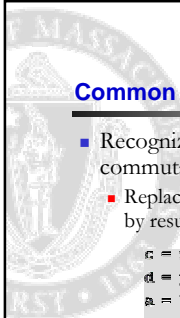
→

```

i = 2;
j = 4;
k = 4;
  
```

- Can we eliminate equivalent expressions *without* constants?

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



Common Subexpression Elimination

- Recognizes textually identical (or commutative) redundant computations
 - Replaces second computation by result of the first

```

c = x + y;
d = y + x;
a = b + c;
e = b + d;
  
```

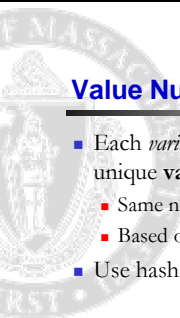
→

```

t1 = x + y;
c = t1;
d = t1;
t2 = b + c;
a = t2;
e = t2;
  
```

- How do we do this efficiently?

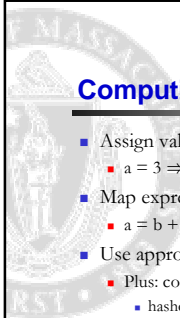
UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



Value Numbering

- Each *variable, expression, and constant*: unique **value number**
 - Same number ⇒ computes same value
 - Based on information from within block
- Use hash functions to compute these

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science



Computing Value Numbers

- Assign values to variables
 - $a = 3 \Rightarrow \text{value}(a) = 3$
- Map expressions to values
 - $a = b + 2 \Rightarrow \text{value}(a) = \text{hash}(+, \text{value}(b), 2)$
- Use appropriate hash function
 - Plus: commutative
 - $\text{hash}(+, \text{value}(b), 2) = \text{hash}(+, 2, \text{value}(b))$
 - Minus: not commutative
 - $\text{hash}(-, \text{value}(b), 2) \neq \text{hash}(-, 2, \text{value}(b))$

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science

Value Numbering Summary

- Forward symbolic execution of basic block
- Each new value assigned to temporary
 - Preserves value for later use even if original variable rewritten
 - $a = x+y; a = a+z; b = x+y$
 $\Rightarrow a = x+y; t = a; a = a+z; b = t;$
- Maps
 - Var to Val
 - specifies symbolic value for each variable
 - Exp to Val
 - specifies value of each evaluated expression
 - Exp to Tmp
 - specifies tmp that holds value of each evaluated expression

Map Usage

- Var to Val
 - Used to compute symbolic value of y and z when processing statement of form $x = y + z$
- Exp to Tmp
 - Used to determine which temp to use if $\text{value}(y) + \text{value}(z)$ previously computed when processing statement of form $x = y + z$
- Exp to Val
 - Used to update Var to Val when
 - processing statement of the form $x = y + z$, and
 - $\text{value}(y) + \text{value}(z)$ previously computed

Computing Value Numbers, Example

<p><i>Original Basic Block</i></p> <pre>a = x+y b = a+z b = b+y c = a+z</pre>	<p><i>New Basic Block</i></p> <pre>a = x+y t1 = a b = a+z t2 = b b = b+y t3 = b c = t2</pre>	
<p><i>Var→Val</i></p> <pre>x → v1 y → v2 a → v3 z → v4 b → v6 c → v5</pre>	<p><i>Exp→Val</i></p> <pre>v1+v2 → v3 v3+v4 → v5 v5+v2 → v6</pre>	<p><i>Exp→Tmp</i></p> <pre>v1+v2 → t1 v3+v4 → t2 v5+v2 → t3</pre>

Interesting Properties

- Finds common subexpressions even if they use different variables in expressions
 - $y = a+b; x = b; z = a+x$
 $\Rightarrow y = a+b; t = y; x = b; z = t$
- Finds common subexpressions even if variable that originally held the value was overwritten
 - $y = a+b; x = b; y = 1; z = a+x$
 $\Rightarrow y = a+b; t = y; x = b; y = 1; z = t$

Problems

- Algorithm has a temporary for each new value
 - $a = x+y; t1 = a;$
- Introduces
 - lots of temporaries
 - lots of copy statements to temporaries
- In many cases, temporaries and copy statements are unnecessary
 - Eliminate with copy propagation and dead code elimination

Global CSE

- Value numbering eliminates some subexpressions but not all

```
read(i);
l = 2 * i;
if (i > 0) goto L1;
j = 2 * i;
goto L2;
L1: k = 2 * i;
L2:
```

- l 's value is not always equal to j 's or k 's value

Available Expressions

- Global CSE requires computation of **available expressions** for blocks b:
 - Expressions on every path in cfg from entry to b
 - No operand in expression redefined
- Then use appropriate temp variable for used available expressions

```

read(i);
l = 2 * i;
if (i > 0) goto L1;
j = 2 * i;
goto L2;
L1: k = 2 * i;
L2:

```

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science 13

Available Expressions: Dataflow Equations

- For a block b:
 - AEin(b) = expressions available on entry to b
 - KILL(b) = expressions killed in b
 - EVAL(b) = expressions defined in b and not subsequently killed in b

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science 14

Available Expressions, Example

```

c = a + b;
d = a * c;
e = d + d;
i = 1;
do {
  f = a + b;
  c = c + 2;
  if (c > 10) {
    k = a + c;
  } else {
    g = d + d;
  }
  i = i + 1;
} while (i > 10);

```

- Build control-flow graph
- Solve dataflow problem
 - Initialize AEin(i) = universal set of expressions
 - $AEin(b) = \bigcap_{j \in Pred(i)} AEout(j)$
 - $AEout(b) = EVAL(i) \cup (AEin(i) - KILL(i))$

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science 15

Next Time

- Partial Redundancy Elimination
- Read ACDI:
 - Ch. 13

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science 16

Value Numbering Example

- Step 1: insert temps for conditionals

```

a = x or y;
b = x or y;
if !z goto L1;
x = !z;
c = x and y;
if (x and y) goto L2;

```

→

```

a = x or y;
b = x or y;
t = !z;
if (t1) goto L1;
x = !z;
c = x and y;
t2 = x and y;
if (t2) goto L2;

```

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science 17

Value Numbering Example

- Step 2:
 - Add entries for each rhs
 - Remove entries when dependent variable changes

statement	hash value
1: a = x or y;	
2: b = x or y;	
3: t = !z;	
4: if (t1) goto L1;	
5: x = !z;	
6: c = x and y;	
7: t2 = x and y;	
8: if (t2) goto L2;	

UNIVERSITY OF MASSACHUSETTS, AMHERST • Department of Computer Science 18