Common Subexpression Elimination

Replace recomputation of expression by use of temp which holds value

Ex. (s1) y := a + b ⟶ Ex. (s1) temp := a + b
(s1') y := temp

(s2) z := a + b (s2) z := temp

Illegal?

How different from value numbering?

Ex. (s1) read(i)
(s2) j := i + 1
(s3) k := i
(s4) l := k + 1

i + 1, k+1
no cse, same value number

Why need temp?

Local and Global

Local CSE (BB)

Before		After
Ex. (s1) c := a + b	⟶	(s1) t1 := a + b
(s2) d := m&n		(s1') c := t1
(s3) e := a + b		(s2) d := m&n
(s4) m := 5		(s3) e := t1
(s5) if(m&n) ...		(s4) m := 5
		(s5) if(m&n) ...
5 instr, 4 ops, 7 vars		6 instr, 3 ops, 8 vars

Always better?

CSE Hash Table

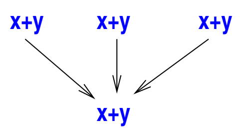
(+, a, b)
(&,m, n)

Global CSE

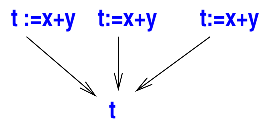
An expression e is ***available*** at entry to B if on every path p from Entry to B, there is an evaluation of e at B' on p whose values are not redefined between B' and B.

Solve by:

1. ***Find Available Expressions (Data flow problem)***
2. ***For each available expression e***



Do backward search from e in CFG to find the evaluations of e



Create new temp t to hold previous evaluations, and replace e by t

Solving Available Expressions

Gen(B) set of expressions evaluated in B available on exit from B

Kill(B) set of expressions killed by B

```
c := a + b
d := w & z
a :=  5
e := e - 7
```

Gen(B) = {w&z}

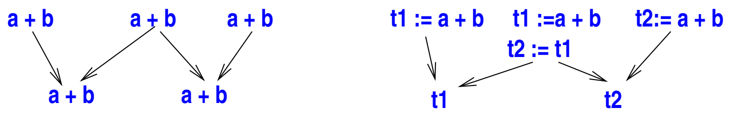
Kill(B) = {all expr. involving c,d,a,e}

Forward or Backward?

$$AEin(B) = \bigcap_{p \text{ pred of } B} AEout(p)$$

$$AEout(B) = Gen(B) \bigcup (AEin(B) - Kill(B))$$

Applying CSE



Create new temps for each occurrence of expression--necessary?

Is CSE always desirable?

When should CSE be applied?

Forward Substitution: Inverse of CSE

Replace copy by reevaluation of expression

Legal?

Loop-invariant Code Motion

Move computations that do not vary in loop outside of loop (pre-header)

Ex.
```
do i := 1, 100
    l := i + (c * 5)
    do j := 1, 100
        a(i.j) := 100*c  + 10*i +j
    enddo
enddo
```

```
t1 :=  (100*c)
t2 :=  (c * 5)
do i := 1, 100
    l := i + t2
    t2 := t1 + 10 * i
    do j := 1, 100
        a(i.j) := t2 +j
    enddo
enddo
```

Almost always a good idea

Especially useful on addressing code

Finding Loop-invariant Code

Find loops

From inner to outer loop order, repeat till no change:

Mark all constant operands as loop-invar.

Mark all operands with all its defs from outside loop-inv.

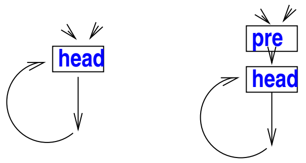
Mark all expressions whose operands are loop-inv. as loop-inv.

Mark all instructions whose only defs are loop -inv. as loop-inv.

Performing Code Motion

Move loop-invariant code to pre-header of loop

May have to create pre-header block



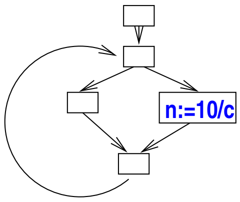
Problems:

Conditional Code

Procedure calls

Problems with Code Motion

Conditional Code



Fix:

1. ***Don't move asst unless***
   asst. dominates all uses, and
   asst dominates all exits of loop

2. ***Move with conditional, if possible***

Problems with Code Motion

Procedure Calls

y:= rand()

Value may be different each call, even if arguments the same

Fix:

1. Don't move, or

2. Interprocedural analysis