Advanced Compilers

CMPSCI 710
Spring 2003
Partial Redundancy Elimination

Emery Berger
University of Massachusetts, Amherst

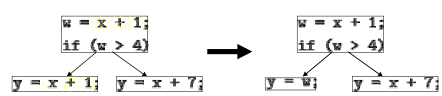
Topics

- Last time
  - Common subexpression elimination
    - Value numbering
    - Global CSE
- This time
  - Partial redundancy elimination

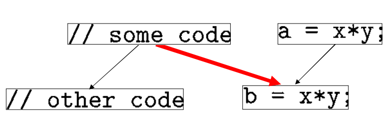
Partial Redundancy

- Partial redundancy:
  - Expression computed more than once on *some* path through control-flow graph
- Partial-redundancy elimination (PRE):
  - Minimizes partial redundancies
    - Inserts and deletes computations (adds temps)
  - Each path contains no more (usually fewer) occurrences of any computation than before
- Dominates global CSE & loop-invariant code motion

PRE Example

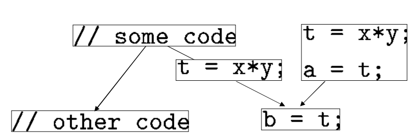


PRE: Problem



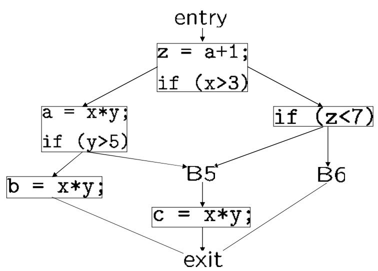
- **Critical edge** prevents redundancy elimination
  - Connects node with two or more successors to one with two or more predecessors
- Why is it a problem?

PRE: Solution

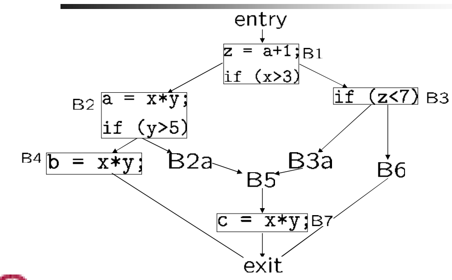


- Split critical edges!
  - Insert empty basic blocks
  - Allows PRE to continue

Big Example: Critical Edges



Big Example: Critical Edges Removed



PRE Dataflow Equations

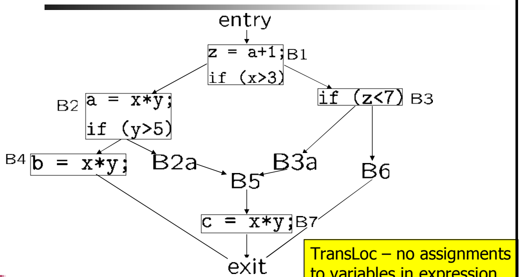
- First formulation [Morel & Renvoise 79] **bidirectional** dataflow analysis
  - Ugly
- This version [Knoop et al. 92]
  - Based on "lazy code motion"
    - Places computations as late as possible
    - Same reductions as classic algorithm
    - Minimizes register pressure
  - Most complex dataflow problem we've ever seen…

Step 1: Local Transparency

- Expression's value is **locally transparent** in a basic block if
  - No assignments to variables that occur in expression
- Set of locally transparent expressions:
  TRANSloc(i)

- Note: Ignore expressions in branches

Local Transparency

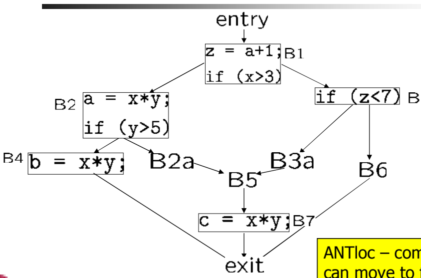


Step 2: Locally Anticipatable

- Expression is **locally anticipatable** in basic block if
  - There is computation of expression in block
  - Moving to beginning of block has no effect
    - No uses of expression nor assignments of variable in block ahead of computation

- Set of locally anticipatable expressions:
  ANTloc(i)

Locally Anticipatable



ANTloc – computes expr, can move to front

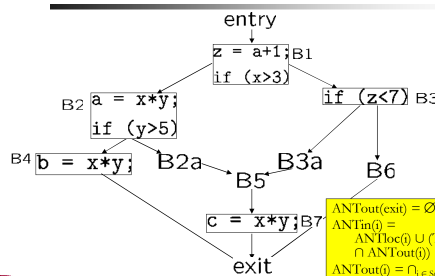
Step 3: Globally Anticipatable

- Expression's value **globally anticipatable** on entry to basic block if
  - Every path from that point includes computation of expression
  - Expression yields same value all along path
- Set of globally anticipatable expressions: ANTin(i)

Globally Anticipatable Expressions: Dataflow Equations

- ANTout(exit) = ∅
- ANTin(i) = ANTloc(i) ∪ (TRANSloc(i) ∩ ANTout(i))
- ANTout(i) = $\cap_{j \in Succ(i)}$ ANTin(j)
- What's the analysis direction?

Globally Anticipatable

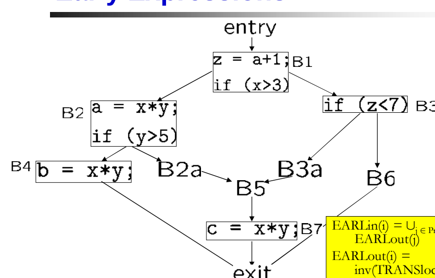


ANTout(exit) = ∅
ANTin(i) = ANTloc(i) ∪ (TRANSloc(i) ∩ ANTout(i))
ANTout(i) = $\cap_{j \in Succ(i)}$ ANTin(j)

Step 4: Earliest Expressions

- Expression is **earliest** at entrance to block if
  - No block from entry to block both:
    - Evaluates expression and
    - Produces same value as at entrance to block
- Defined in terms of local transparency and globally anticipatable expressions
  - EARLin(i) = $\cup_{j \in Pred(i)}$ EARLout(j)
  - EARLout(i) = inv(TRANSloc(i)) ∪ (inv(ANTin(i)) ∩ EARLin(i))
  - Initialize EARLin(entry) = U_{exp}

Early Expressions

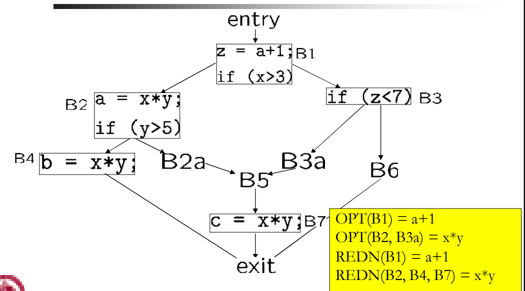


EARLin(i) = $\cup_{j \in Pred(i)}$ EARLout(j)
EARLout(i) = inv(TRANSloc(i)) ∪ (inv(ANTin(i)) ∩ EARLin(i))

PRE Transformation

- We'll cut to the chase:
  - Latest, Isolated expressions
    - Use earliest, globally anticipatable
  - OPT(i) = latest but not isolated = LATEin(i) ∩ inv(ISOLout(i))
  - REDN(i) = used but not optimal = ANTloc(i) ∩ inv(LATEin(i) ∪ ISOLout(i))
- Insert fresh temporaries for OPT expressions, replace uses in REDN

OPT, REDN, PRE



Conclusion

- PRE
  - Subsumes global CSE & loop-invariant code motion
  - Complex (but unidirectional) dataflow analysis problem
  - Can only reduce number of computations and register pressure
- Next time
  - Register allocation: ACDI ch.16, pp. 481-524