Partial Redundancy Elimination

CS243 Review Session
Full Redundancy

\[ \begin{align*} x &= b + c \\ y &= b + c \\ z &= b + c \end{align*} \]
Partial Redundancy

\[ x = b + c \]

\[ z = b + c \]
Partial Redundancy

\[ t = b + c \]
\[ x = t \]
\[ z = t \]
\[ t = b + c \]
\[ u = a + b \]
\[ v = a + b \]
\[ w = a + b \]

\[ b = \text{read()} \]
Add blocks on critical edges
\[ u = a + b \]
\[ v = a + b \]
\[ w = a + b \]

Anticipated expressions:
places where it is safe to place \( a + b \)
u = a + b
v = a + b
w = a + b

b = read()

Can delete added blocks where \( a + b \) is not anticipated
Available expressions: points where $a + b$ could be made available
Earliest: when can we earliest compute $a + b$
Earliest: when can we earliest compute \( a + b \)
How much can we postpone evaluating $a + b$?
Latest: need to compute $a + b$ here
u = a + b
v = a + b
w = a + b

b = read()

Latest: need to compute $a + b$ here
u = a + b

v = a + b

w = a + b

b = read()

Remove added blocks where we are not going to compute anything
\[ u = t \]
\[ t = a + b \]
\[ v = t \]

Use a temporary variable to store the result

\[ t = a + b \]
\[ t = a + b \]
\[ b = \text{read()} \]
\[ t = a + b \]
\[ u = t \]

\[ t = a + b \]

\[ v = t \]

\[ t = a + b \]

\[ w = t \]

\[ b = \text{read()} \]

\[ t = a + b \]
\[
\begin{align*}
    u & = t \\
    t & = a + b \\
    v & = t \\
    t & = a + b \\
    w & = t \\
    b & = \text{read()}
\end{align*}
\]

Result not used beyond the block in which the variable is defined.
u = t
v = a + b
t = a + b
w = t

Clean up unrequired temporaries

b = read()
\[ u = t \]
\[ v = a + b \]
\[ t = a + b \]
\[ w = t \]
\[ b = \text{read()} \]
More Examples
\[ i = 0 \]

\[ a = b + c \]

\[ i = i + 1 \]

\[ i < 1000 \]

\[ z = b + c \]
\[
i = 0
\]
\[
a = b + c
\]
\[
i = i + 1
\]
\[
i < 1000
\]
\[
z = b + c
\]
c = 2

B2

B3

B4

c = 2

B1

B5

B6

a = b + c

B7

d = b + c

B8

B9

e = b + c

B10

B11
B1

B2

B3

B4

B5

B6

B7

B8

B9

B10

c = 2

a = b + c

d = b + c

e = b + c

B1

B2

B3

B4

B5

B6

B7

B8

B9

B10

t = b + c

a = t

d = t

e = t
Dominators

CS243 Review Session
Example

Draw the dominator tree for this control flow graph.
Draw the dominator tree for this control flow graph.
Example

IN = RED
OUT = BLACK

Draw the dominator tree for this control flow graph.
Aside: there are algorithms for constructing the dominator tree directly
- Tarjan’s algorithm (based on DFS)
- Buchsbaum’s algorithm
Example

IN = RED
OUT = BLACK

Find the back edges and natural loops in this graph.
Example

Find the back edges and natural loops in this graph.

Dominator Tree
Example

IN = RED
OUT = BLACK
BACK EDGE = ORANGE

Find the back edges and natural loops in this graph.

Natural loop for back edge $K \rightarrow M$: all nodes
* All nodes can reach $K$ without passing through $M$
Post-dominators

How would we compute the post-dominators for this graph?
Definitions

A block B dominates block B’ if every path from the entry to B’ goes through B.

A block B postdominates block B’ if every path from B’ to the exit of the graph goes through B.

If B dominates B’ and B’ postdominates B, B and B’ are control equivalent.
* One is executed when and only when the other is
Example

if (a == 0) goto L

L:

c = b

e = d + d
Example

1. B1 and B3 are control equivalent.
Code Motion

If two blocks are control-equivalent, you may move instructions between the two (upward/downward code motion) assuming there are no conflicting data dependences

More to come next week: instruction scheduling lecture
SSA

Construction of the static single assignment form (SSA) requires dominance frontier information.

The dominance frontier of a node \( d \) is the set of all nodes \( n \) such that \( d \) dominates an immediate predecessor of \( n \), but \( d \) does not strictly dominate \( n \).

More to come in Homework 3: converting to SSA form