263-2810: Advanced Compiler Design

3.3 Dominance frontier

Thomas R. Gross

Computer Science Department
ETH Zurich, Switzerland
Outline

- 3.1 Graphs
- 3.2 Approaches to insertion of $\phi$ nodes
- 3.3 Dominance frontier
- 3.4 Algorithm for insertion of $\phi$ nodes
- 3.5 Algorithm for variable renaming
- 3.6 Example
\[DF(Q) = \{ N \mid \exists \text{ path from } Q \text{ to } N, \]

\[\neg(Q \gg N),\]

\[\exists \text{ predecessor } P \text{ of } N \text{ s.t. } Q \gg P \}\]
\[ \text{DF}_{\text{UP}}(Y) \]

- A node \( N \in \text{DF(child)} \) is \( \in \text{DF(parent)} \) if \( \neg (\text{parent} \gg N) \)
- \( \text{DF}(Y) = \{ Z_1, Z_2 \}, \text{DF}(X) = \{ Z_1 \} \)

\[ \text{DF}_{\text{UP}}(Y) = \{ N \mid N \in \text{DF}(Y) \text{ and } \neg (\text{Parent}(Y) \gg N) \} \]
- \( \text{Parent}(Y) = \text{idom}(Y) \)
Computing $DF(X)$

- $DF(Y) = \{ Z_1, Z_2 \}$
- $DF_{UP}(Y) = \{ Z_1, Z_2 \}$
- $DF(Y) = \{ Z_1, Z_2 \}$
- $DF_{UP}(Y) = \{ Z_1 \}$
$DF_{\text{Local}}(X)$

- $DF_{\text{Local}}(X)$ is the set of nodes $N$ that are direct successors of $X$ but not descendants

- $DF_{\text{Local}}(Y) = \{ Z_1, Z_2 \}$

- $DF_{\text{Local}}(X) = \{ Z_3 \}$
Only direct successors contribute to $DF_{\text{Local}}(X)$

- Descendants $D_i$ (that are children) contribute through $DF_{UP}(D_i)$
Incremental computation of DF(X)

- \( \text{DF} (X) = \text{DF}_{\text{Local}}(X) \cup \bigcup \text{DF}_{\text{UP}}(Y_i) \text{ with } Y_i \text{ child of } X \)

- \( \text{DF}_{\text{Local}}(X) \) : Set of nodes in DF(X) determined by (CFG) successors

- \( \text{DF}_{\text{UP}}(Y) \) : Set of nodes in DF(X) contributed by children Y of X
- Efficient computation
- Compute $DF_{\text{Local}}(X)$: inspect all direct CFG successors
  - Usually a small number
- Assume $DF(Y)$ has been computed for all children $Y_i$ of $X$
- Get $DF_{\text{up}}(Y_i)$: all nodes $N \subseteq DF(Y_i)$ with $\neg (X \gg N)$
Computing $DF_{\text{Local}}(X)$

$DF_{\text{Local}}(X) = \{ N \mid N \text{ is a direct successor of } X,$
\[ \neg ( X \triangleright N) \]
\[ \exists N' \text{ s. t. } X \triangleright N' \text{ and } N' \text{ predecessor of } N \}$

- Really easy if $X$ is a leaf node of the DT
  - No children in dominator tree $\implies$ only direct successors must be inspected
DF for leafs

- $\text{DF} (X) = \text{DF}_{\text{Local}}(X) \cup \left( \bigcup ( \text{DF}_{\text{UP}}(Y_i) \text{ with } Y_i \text{ child of } X) \right)$
- Leaf nodes have no children
- $X$ leaf node: $\text{DF} (X) = \text{DF}_{\text{Local}}(X)$
3.3.3 Putting $DF_{Local}$ and $DF_{UP}$ together

- $DF(X) = DF_{Local}(X) \cup \bigcup (DF_{UP}(Y_i) \text{ with } Y_i \text{ child of } X)$

- Must combine $DF_{Local}$ and $DF_{UP}$ of children
Visit nodes N in a bottom-up traversal of the DT
for each node N {
    DF(N) = Ø
    for each node X, X successor of N {
        if ( idom(X) ≠ N) { DF(N) = DF(N) ∪ { X } }
    }
    for each node Z, Z child of N { 
        for each Y ∈ DF(Z) {
            if ( idom(Y) ≠ N ) { DF(N) = DF(N) ∪ { Y } }
        }
    }
}
3.3.4 Example

\[ D^F(4) = \{ 8 \} \]
\[ D^f(5) = \{ 8 \} \]
\[ D^f(6) = \{ 10 \} \]
\[ D^f(7) = \emptyset \]
\[ D^f(9) = \emptyset \]
\[ D^f(2) = \emptyset \]
\[ D^F(2) = \{ 10 \} \]
\[ D^F(5) = \{ 8, 9 \} \]

\[ D^f(3) = \{ 7, 9 \} \]

\[ D^f(10) = \emptyset \]

Entry: \( \{ 8, 7, 8, 3, 7, 6, 3, 10 \} \)

Exit: \( \{ 2, 3, 10, 4, 5, 8, 6, 7, 9 \} \)
Example 2
3.4 Algorithm for insertion of $\phi$ nodes

- Given a CFG, DF for all nodes (blocks)
- **For each variable** $V$, we need
  - List of all nodes that contain definitions of $V$
  - (Basic blocks with assignments to $V$)
  - Call this list ASSIGN($V$)
- **Idea:** insert $\phi$ functions for $V$ for all CFG nodes in ASSIGN($V$)
  - Check if inserting a $\phi$ function (a special assignment, creates new version) requires insertion of additional $\phi$ functions
- $\text{has}_\phi\_\text{fct}(X)$: true for a block $X$ in CFG iff $X$ contains a $\phi$ function for $V$
- **worklist** $W$: set of blocks (nodes) still to be processed
- **added_to_worklist($X$)**: true for block $X$ iff has been added to $W$
Implementation concerns

- Need to keep track where a $\phi$ function is inserted
- One option: bit vector
  - Length: number of basic blocks (nodes in CFG)
- Drawback: potentially inefficient

- Better idea: use integer (counter) to keep track of processing CFG nodes
  - Record when a $\phi$ function is inserted
  - $\text{has}_\phi\_\text{fct}(X) = C$ with $C$ integer, $C$ iteration when $\phi$ function is inserted
  - $\text{added}_\text{to}_\text{worklist}(X) = C$ with $C$ integer, $C$ iteration when added to $W$
int count = 0
for each node X in CFG {
    has_\_\_\_fct(X) = count
    added_to_worklist(X) = count
}
for all variables V {
    count ++
    W = ASSIGN(V)
    for all X ∈ ASSIGN(V) { added_to_worklist(X) = count }
    while ( W ≠ ∅ ) {
        pick B from W, remove B from W
        for all nodes Y ∈ DF(B) {
            if ( has_φ_fct(Y) < count ) {
                insert φ function into Y
                has_φ_fct(Y) = count
                if (added_to_worklist(Y) < count) { add Y to W, added_to_worklist(Y) = count }
            } // if no φ_fct
        } // for all nodes in DF
    } // while
} // for all variables
Example

ENTRY

a = b =

b =

EXIT

EXIT

ENTRY

a = b =

b =

ENTRY

a = b =

b =

ENTRY

a = b =
Example

ENTRY

0

a =
b =

1

a = φ( , )
b = φ( , )

2

b =

3

b = φ( , )

4

5

a = φ( , )
b = φ( , )

6

7

8

a = φ( , )
b = φ( , )

EXIT
3.5 Renaming variables

- Given a CFG, with $\phi$ functions inserted
- Given a statement $S$ in block $B$ of the CFG, find version(s) for all variables $V$
  - on the RHS (or in an expression)
  - on the LHS
  - in a $\phi$ function
Possible setup

$A = \phi(\ldots)$

$A_1 = \ldots$

$A_2 = \ldots$

$B$

$C$

donimated by $B$

use version set in $B$

donimated by $C$

use version set in $C$
Pick ... version

- Inside region dominated by B
  - Use version assigned in B
  - Unless there is a more recent version
    - Inside region dominated by C ......
  - and so on

- Process nodes of the CFG in the order defined by DT

- Assume C is processed before A
  - Make sure to use In A version defined in B
  - → restore old set of versions when done with C
    - and its descendent
Stack of versions

- For each variable \( V \): stack of versions
  
  \[
  \text{stack}[V]
  \]
  
  Initially empty

- For each variable: counter
  
  \[
  \text{counter}[V]
  \]
  
  Initialized to 0
Process (basic block X) {
  for all statements S in X {
    if ( S is not a $\phi$ function) {
      for each variable V in RHS {
        replace V with $V_i = \text{Stack}[V].\text{top}$
      }  // RHS
      for each variable V on the LHS {
        $c = \text{counter}[V]$
        $\text{Stack}[V].\text{push}( c )$
        replace V with $V_c$ on the LHS
        $\text{counter}[V] = c + 1$
      }  // LHS
    } // for all statements
  } // for all statements
for all successors Y of X {
  for all φ functions F in Y {
    let V be the LHS of F
    let X be the $k$-th predecessor of Y
    set the $k$-th argument of F to $V_i = \text{Stack}[V].\text{top}$
  } // for all φ functions
} // for all successors
for all children C of X {
    process ( C )
}
for all assignments A in X {
    if V is set by A {
        Stack[V].pop
    }
} // for all assignments
}
} // for all blocks
Example

ENTRY

0

\[ a = b = \]

1

\[ a = \phi(\text{ }, \text{ }) \]
\[ b = \phi(\text{ }, \text{ }) \]

2

\[ b = \]

3

\[ a = \phi(\text{ }, \text{ }) \]
\[ b = \phi(\text{ }, \text{ }) \]

4

\[ a = \]

5

\[ a = \phi(\text{ }, \text{ }) \]
\[ b = \phi(\text{ }, \text{ }) \]

6

\[ b = \]

7

\[ b = \phi(\text{ }, \text{ }) \]

8

\[ b = \phi(\text{ }, \text{ }) \]

9

\[ a = \phi(\text{ }, \text{ }) \]
\[ b = \phi(\text{ }, \text{ }) \]

EXIT