263-2810: Advanced Compiler Design

3.0 SSA form for arbitrary control flow graphs

Thomas R. Gross

Computer Science Department
ETH Zurich, Switzerland
SSA format

- φ functions allows us to deal with a basic block that has multiple predecessors

- To turn a program into SSA format
  - Insert φ nodes
  - Determine version to be used as operand
  - Produce new versions when there is an assignment
    - Or a φ function delivers the correct version

- So far: syntactic approach
  - Works for well-structured programs
  - Next: an approach for general control flow graphs
  - But first: review of concepts for graphs
Outline

- 3.1 Graphs
- 3.2 Approaches to insertion of φ nodes
- 3.3 Dominance frontier
- 3.4 Algorithm for insertion of φ nodes
- 3.5 Algorithm for variable renaming
- 3.6 Example
3.1 Graphs

- **Control flow graph: central data structure**
  - Nodes: basic blocks (or sequences of straight-line programs)
  - Edges: directed, *possible control flow*

- **Special nodes**
  - ENTRY: special without predecessor
  - EXIT: special without successor

- Edge from ENTRY to EXIT
Dominance

- Given a CFG. A node A (or basic block A) dominates node B (basic block B) if A is on every path from ENTRY to B.

A \textit{dom} B

A \textit{dom} A

X \sim \textit{dom} Y : X does not dominate Y
Example 1

ENTRY

BB0

BB1

BB2

BB3

BB0 dom BB1 dom BB2 dom BB2

BB0 dom BB1

BB0 dom BB2

BB0 dom BB3

BB1 dom BB1
Example 2

ENTRY

A

B

C

D

EXIT
Immediate dominator

- $A \text{idom} B$
  - $A \text{dom} B$
  - $\exists X$ such that $A \text{dom} X$ and $X \text{dom} B$ ($X \neq A, X \neq B$)

ENTRY $A$ idom $A$

$A$ idom $B$

$B$ idom $C$

$B$ idom $D$

ABCBD, OK

A idom D2

No

ABD

ABD, last idom

A-B-D A-C-D
Example 2

ENTRY

A

B

C

D

EXIT

ENTRY idom A
A idom B
B idom C
B idom D

ABCBD

OK

A idom D

A

B

C

D

A idom D

A idom D

A

B

C

D

A idom D

A

B

C

D

A idom D

A-B-D

A-C-D

A-B-D

A-C-D

A-B-D

A-C-D
Dominator tree

- The dominator tree (DT) captures the *immediate dominator* relationship
  - X idom Y: edge $X \rightarrow Y$ in DT

- There is exactly one immediate dominator
  - That’s why it’s a tree ...
  - Last dominator on any path from ENTRY
Terminology

- **Control flow graph**
  - predecessor
  - successor
  - direct (predecessor/successor): There is an edge --- indicates possible control transfer

- **Example**
  - B direct successor of A
  - C direct successor of B
  - A direct predecessor of B
  - D successor of A
  - A predecessor of C
  - C *possible* predecessor of B
  
  ENTRY → A → B → C → B ..... vs ENTRY → A → B → D → EXIT
Terminology 2

- **Dominator tree**
  - child
  - parent
  - ancestor: parent, grandparent, great-grandparent,...
  - descendant: child, grandchild, great-grandchild, ...

- **Example**
  - A parent of B
  - B child of A
  - C child of B

- No statement about actual control flow
- ENTRY → A → B → D → EXIT
Dominator relationship

- *dom* reflexive
  - $X \text{ dom } X$

- Sometimes we want an *irreflexive (anti-reflexive)* relationship
  - Want to be sure that $A \text{ dom } B$ implies $A \neq B$

- $X$ strictly dominates $Y$: $X \text{ dom } Y$ and $X \neq Y$
  - $X \gg Y$

- Distinguish from (weak) domination: $X \text{ dom } Y$
  - $X \gg Y$
Computing dominator relationship

data flow equation

\[ \text{DOM}(\text{ENTRY}) = \{\text{ENTRY}\} \]

set of dominating nodes

\[ \text{DOM}(N) = \bigcup_{\text{predecessor P_i of N}} \text{DOM}(P_i) \]

initialization

\[ \text{DOM}(\text{ENTRY}) = \{\text{ENTRY}\} \]

\[ \text{DOM}(N) = \bigcup_{N \neq \text{ENTRY}} \text{DOM}(N) \]

iterate until no more change

for all nodes Q, Q \neq \text{ENTRY}

\[ \text{DOM}(Q) = \{Q\} \cup \bigcup_{P_i \text{ P_i predecessor of N}} \text{DOM}(P_i) \]
3.2 Insertion of $\phi$ nodes

- Where could we insert $\phi$ nodes?
- Where should we insert $\phi$ nodes?
Example graph
- Program not well-structured

- Insert $\phi$ nodes – Option A:
  - Insert a $\phi$ node into the block that uses a variable
  - Example: $x$ is used in basic block BB6
Example graph

ENTRY

EXIT

\[ X = \phi(x, y, z) \]

BB6

= X
Insert $\phi$ nodes – Option B:

- Insert a $\phi$ node as early as possible
- Even if there is no use of a variable in basic block
- Example: nodes in BB3, BB5, and BB6
Example graph

\[
ENTRY \quad x = \phi(x) \\
\quad x = \phi(x) \quad BB3 \\
\quad x = \phi(x) \quad BB5 \\
\quad x = \phi(x) \quad = x \quad BB6 \\
EXIT
\]
Option B: Number of arguments to $\phi$ function directly depends on the number of predecessors in CFG

- Fixed for a given programming language resp. implementation
- case/switch statement may result in unlimited number of predecessors unless compiled into cascading set of if-statements

Option A: Number of arguments to $\phi$ function depends on number of paths that reach a given point

- Unbounded

Option B preferred by compiler designers
Setup

- Let us assume there is an initial (dedicated) assignment to each variable
  - For X, Y, Z,... we have $X_0$, $Y_0$, $Z_0$, ...
  - “Placed” into the ENTRY node
  - “pseudo-assignment”

- Benefits
  - There is at least one assignment on every path
  - If the RHS of an assignment or expression reads $X_0$ there is possibly a use of an uninitialized variable
CFG nodes with $\phi$ functions

- Consider two nodes A, B
- A defines $X$, B uses $X$
- No function is needed along path from A to B if $A \gg B$
- But $A = B$ is possible. Need to deal with

- Extend definition of dominance from basic blocks (nodes of CFG) to operations in basic block
  - definitions
  - uses
- Definition (def) $d_i$ in block A
- Use (use) $u_j$ in block B
  \[ d_i \implies u_j \]
- iff $d_i$ is in block A, $u_j$ is in block B, and $d_i$ is the last definition along a path from A to B (i.e., from $d_i$ to $u_j$)
- Can define $d \implies u_j$ as well
\(\neg (d_i \gg u_j)\)

- \(\neg (d_i \gg u_j)\): \(d_i\) does not dominate \(u_j\)

- \(\exists\) at least one path from \(\text{ENTRY}\) to \(u_j\) that does not include \(d_i\) (as the last definition of \(X\))
Detailed look

- No $\phi$ function if $d \gg u$
- $\phi$ function if no dominance
The diagram illustrates a network with nodes labeled as follows:

- Node $d_1$ dominates node $d_2$.
- Node $d_2$ dominates node $d_3$.
- Node $d_1$ dominates node $n_1$.
- Node $d_2$ dominates node $n_2$.
- Node $d_3$ dominates node $n_3$.

The text below the diagram reads:

- Good places for markers at node $n_1$.
- Set dominated by $d_1, d_2, d_3$.

The diagram also includes arrows indicating dominance relationships between nodes.
Placement of $\phi$ functions

- Look for nodes $n_1$, $n_2$, ... that have these properties
  - $n_1$, $n_2$, .... are not dominated by definitions $d_1$, $d_2$, $d_3$, ...
  - predecessors of $n_1$, $n_2$, ... are dominated
  - $n_1$, $n_2$, ... on a path from $d_1$, $d_2$, ... to $u$

- These nodes form the *dominance frontier*

- Note: $u$ may be $n_1$, $n_2$, ...