Another Form of Data-Flow Analysis

Propagation of values

- for a variable reference, where is the value produced?
- for a variable definition, where is the value consumed?

Possible answers

- reaching definitions, live variables
- def-use, use-def chains
- static single assignment (SSA)

Def-Use and Use-Def Chains

- directly connect producers and consumers
- extend reaching definitions to add edges
- extend live variables to track uses, add edges
- advantage: bypass intervening flow graph
- disadvantage: requires more space
Static Single Assignment Form

What is SSA?

- each assignment to a variable is given a unique name
- all of the uses reached by that assignment are renamed

Example

\[
\begin{align*}
V & \leftarrow 4 & V_0 & \leftarrow 4 \\
& \leftarrow V + 5 & & \leftarrow V_0 + 5 \\
V & \leftarrow 6 & V_1 & \leftarrow 6 \\
& \leftarrow V + 7 & & \leftarrow V_1 + 7
\end{align*}
\]

Why is this useful?

- representation explicitly connects definitions to their uses (and vice versa)
- more compact representation than def-use and use-def chains
- definitions kept in a separate data structure from CFG
- merging of values is explicit
Handling Multiple Reaching Definitions

\(\phi\)-functions (aka, \(\phi\)-nodes)

- For some CFG node \(n\), a function of the form
  \[ V_n \leftarrow \phi(V_{p1}, V_{p2}, \ldots), \]
  where each subscripted \(V\) corresponds to the definition reaching this point from \(n\)'s predecessors \(p1\) and \(p2\)

Where do we place \(\phi\)-functions?

**Intuitively.** At the first point where paths in the CFG merge that have distinct definitions for the same variable

**Formally.** If there exist two non-null paths \(X \rightarrow^+ Z\) and \(Y \rightarrow^+ Z\) that converge for the first time at node \(Z\), and nodes \(X\) and \(Y\) contain assignments to \(V\) (in the original program), then a \(\phi\)-function for \(V\) must be inserted at \(Z\) (in the new program)

Placement of \(\phi\)-functions subject to this condition yields “minimal” SSA form
Examples of $\phi$-Functions

B₁ if (...)  

B₂ X ← 5  
B₃ X ← 3  

B₄ Y ← X

B₁ if (...)  

B₂ X₀ ← 5  
B₃ X₁ ← 3  

B₄ X₂ ← $\phi(X₀, X₁)$  
Y ← X₂
Another Example: Loops

B₁ I ← 1

B₂ I ← I + 1

B₁ I₀ ← 1

B₂ I₁ ← φ(I₂, I₀)
I₂ ← I₁ + 1
Computing SSA Form

1. Insert $\phi$-functions
   (a) Build dominator tree
   (b) Compute dominance frontiers and their closure
   (c) Rename variables
2. Translate back from SSA form

The SSA graph is simply the graph built by adding def-use and/or use-def chains to the program in SSA form during SSA construction

R. Cytron et al., “Efficiently computing static single assignment form and the control dependence graph”, ACM Transactions on Programming Languages and Systems (TOPLAS), 13(4), October 1991
Insert \( \phi \)-functions: Dominators

If \( X \) appears on every path from \textbf{Entry} to \( Y \),
then \( X \) dominates \( Y \) \((X \triangleright Y)\)

If \( X \triangleright Y \) and \( X \neq Y \),
then \( X \) strictly dominates \( Y \) \((X \triangleright Y)\)

The \textit{immediate dominator} of \( Y \) \((\text{idom}(Y))\)
  is the closest strict dominator of \( Y \)

\text{idom}(Y) \) is \( Y \)'s parent in the \textit{dominator tree}
Dominance Frontiers

The *dominance frontier* of node $X$ is the set of nodes $Y$ such that

- $X$ dominates a predecessor of $Y$, but
- $X$ does not strictly dominate $Y$

$$DF(X) = \{ Y \mid \exists P \in \text{Pred}(Y), \quad (X \gg P \text{ and } X \gg Y) \}$$

The dominance frontier can be subdivided into two components:

- $DF_{local}(X) \equiv \{ Y \in \text{Succ}(X) \mid X \gg Y \}$
- $DF_{up}(X) \equiv \{ Y \in DF(Z) \mid Z \in \text{Children}(X) \land X \gg Y \}$

Then,

$$DF(X) = DF_{local}(X) \cup DF_{up}(X)$$

$Succ = \text{immediate successors in the CFG}$

$Children = \text{descendents in the dominator tree}$

† *Intuitively, dominance frontier is point just beyond region a node dominates*
for each X in a bottom-up traversal of the dominator tree

\[
\text{DF}(X) \leftarrow \emptyset \\
\text{for each } Y \in \text{Succ}(X) \quad /* \text{local} */
\]
\[
\quad \text{if } \text{idom}(Y) \neq X \text{ then}
\quad \quad \text{DF}(X) \leftarrow \text{DF}(X) \cup \{Y\}
\]

\[
\text{for each } Z \in \text{Children}(X) \quad /* \text{up} */
\]
\[
\text{for each } Y \in \text{DF}(Z)
\]
\[
\quad \text{if } \text{idom}(Y) \neq X \text{ then}
\quad \quad \text{DF}(X) \leftarrow \text{DF}(X) \cup \{Y\}
\]

\[
\text{Succ} = \text{immediate successors in the CFG} \\
\text{Children} = \text{descendents in the dominator tree}
\]
Dominance Frontiers Example

DF(8) = 

DF(9) = 

DF(7) = 

DF({8,9}) = 

DF(10) = 

DF({8,9,10}) =
Dominance Frontier Closure

Extend the dominance frontier mapping from nodes to sets of nodes:
\[ \text{DF}(\mathcal{L}) = \bigcup_{X \in \mathcal{L}} \text{DF}(X) \]

The \textit{iterated} dominance frontier \( \text{DF}^+(\mathcal{L}) \) is the limit of the sequence:
\[ \begin{align*}
\text{DF}_1 &= \text{DF}(\mathcal{L}) \\
\text{DF}_{i+1} &= \text{DF}(\mathcal{L} \cup \text{DF}_i)
\end{align*} \]
(i.e., a closure)

\textit{Theorem}

The set of nodes that need \( \phi \)-nodes for any variable \( V \) is the iterated dominance frontier \( \text{DF}^+(\mathcal{L}) \), where \( \mathcal{L} \) is the set of nodes with assignments to \( V \).
Algorithm For Inserting $\phi$-nodes

for each variable $V$
    HasAlready $\leftarrow \emptyset$
    WorkList $\leftarrow \emptyset$
    for each node $X$ containing an assignment to $V$
        WorkList $\leftarrow$ WorkList $\cup \{X\}$

while WorkList $\neq \emptyset$
    remove $X$ from $W$
    for each $Y \in$ DF($X$)
        if $Y \notin$ HasAlready then
            insert a $\phi$-node for $V$ at $Y$
            HasAlready $\leftarrow$ HasAlready $\cup \{Y\}$
            WorkList $\leftarrow$ WorkList $\cup \{Y\}$
SSA Form: Renaming the Variables

Data structures used for renaming variables

**Stacks** array of stacks, one for each original variable \( V \)
- Contains the subscript of the most recent definition of \( V \)
- Initially, \( \text{Stacks}[V] = \text{EmptyStack}, \forall V \)

**Counters** an array of counters, one for each original variable
- Contains the number of assignments to \( V \) processed
- Initially, \( \text{Counters}[V] = 0, \forall V \)

procedure **GenName**(Variable \( V \))
- \( i \leftarrow \text{Counters}[V] \)
- replace \( V \) by \( V_i \)
- Push \( i \) onto \( \text{Stacks}[V] \)
- \( \text{Counters}[V] \leftarrow i + 1 \)

**Rename** - a recursive procedure

- Walks the control flow graph in preorder
- Initially, call **Rename**(entry)
procedure **Rename**(Block X)
   for each \(\phi\)-node P in X  
   GenName(LHS(P))
   for each statement S in X  
   for each variable \(V \in \text{RHS}(S)\)  
   replace V by \(V_i\), where \(i = \text{Top}(	ext{Stacks}[V])\)  
   for each variable \(V \in \text{LHS}(S)\)  
   GenName(V)
   for each \(Y \in \text{Succ}(X)\)
      \(j \leftarrow \) position in \(Y\)'s \(\phi\)-nodes corresponding to X
      for each \(\phi\)-node P in Y
         replace the \(j^{th}\) operand of \(\text{RHS}(P)\) by \(V_i\)
         where \(i = \text{Top}(	ext{Stacks}[V])\)
   for each \(Z \in \text{Children}(X)\)
      Rename(Z)
   for each \(\phi\)-node or statement S in X
      for each \(V_i \in \text{LHS}(S)\)
      pop Stacks[V]
What Happens To Stacks During Renaming

V ←
...
V ←
...
V ←

Before

Stacks

V

Stacks

V

After

V

CMSC 731/838T  Lecture 13, Page 15
Computing SSA Form

Complete algorithm

- compute the dominance frontiers
- insert $\phi$-nodes
- rename the variables

Theorem

Any program can be put into “minimal” SSA form using this algorithm.

Optimizations: [fewer $\phi$-nodes than “minimal”]

- pruned – eliminate any dead $\phi$-nodes
- semi-pruned – eliminate $\phi$-nodes for variables dead on exit from all basic blocks

Translate back from SSA form

- restore original names to variables
- replace $\phi$-nodes with copies in CFG predecessors
- delete all $\phi$-nodes
Example: Translating From SSA Form

\[
\begin{align*}
B_1 & \text{ if } (...) \\
B_2 & \quad X_0 \leftarrow 5 \\
B_3 & \quad X_1 \leftarrow 3 \\
B_4 & \quad X_2 \leftarrow \phi(X_0, X_1) \\
& \quad \quad Y \leftarrow X_2
\end{align*}
\]